A THEORETICAL AND PRACTICAL INVESTIGATION OF TOOLS AND TECHNIQUES FOR THE STRUCTURING OF DATA AND FOR MODELLING ITS BEHAVIOUR

Volume I of III

by PAUL BARRIE FELDMAN B.Sc(Hons)

a PhD thesis

UNIVERSITY OF WARWICK
COVENTRY, ENGLAND

SCHOOL OF INDUSTRIAL AND BUSINESS STUDIES

September 1986

C O N T E N T S

Title Page

VOLUME II

PART II RESEARCH

Title Page

VOLUME III

PART III CONCLUSIONS

X APPENDICES

LIST OF ILLUSTRATIONS

Chapter C

Chapter D

Chapter E

VOLUME III

Appendices

Appendix X1

Appendix X3

ACKNOWLEDGEMENTS

This research was initially based at Thames Polytechnic under the supervision of Guy Fitzgerald, Brian Knight and Tom Crowe. At the time it was collaborative with CACI Inc International where I had the invaluable aid of their consultants, particularly Tim Bourne, Keith Short, Steve Wasserman and Duncan Walker.

Later the research moved to the The University of Warwick under the supervision of Guy Fitzgerald and Bob Hurrian. It was sponsored here by James Martin Associates where I had the aid of Ian MacDonald, Keith Short, Clive Mabey and John Dodd.

I would like to give grateful thanks here to my main supervisor throughout, Guy Fitzgerald, without whom this project would not have succeeded nearly as well as it has.

I would also like to take this opportunity to thank my wife Lesley for supporting my efforts throughout the long dark days of research and for helping with my English (usually fighting a losing battle).

SOURCE ACKNOWLEDGEMENTS

Many parts of this thesis have been published separately during the research period as part of the research activity (as described in Chapter F this was part of the validation of the research). Some of these publications were joint. All the publications are described in Appendix X5.

Chapter A is completely novel. In Chapter B B1 and B2 are new. B3 has been published before in [FEFI, 85a] and [FEFI, 85b], but in a much less complete form. Those parts which were published are almost all my own work. B4 and B8.2 are new, but B5.1 appeared in [FEFI, 85a] and B5.2 in [FEFI, 85b]. Most of B7 is unpublished, but some was published. B8 and B9 are unpublished.

In Chapter C, C1 and C2 are new. Parts of C3 were published as [FEMI, 85] and [FEMI, 86]. Again, most of those parts used were my own work. C4 is entirely new but C5 appeared in [FEMI, 86]. C6 is novel, whereas part C7 is an expansion of part of [FEMI, 86]. C8 and C9 are new.

In Chapter D, D1 and D2 are new. D3 is extracted and modified from [FMM, 86] and was largely written by me in the original. D4 and D5 are both new.

Chapter E is almost completely all from [FELD, 84], a sole publication.

Chapters F to I are new.

In the appendices: X1 comes from [FMM, 86], but was mostly written by Ian Macdonald, except X1.4 and X1.5 which are mostly my writings. X2 is unpublished as are X5 to X7. X3 on the other hand again comes from [FMM, 86] and was originally produced by Clive Mabey with aid from myself. X4 is an extraction and addition of the James Martin Associates Information Engineering Glossary as these are the terms used in the thesis.

SHORT SUMMARY

This thesis is about data and behaviour modelling for information system development. It has been sponsored at different times by two specialist consultancies: CACI Inc International and James Martin Associates.

Initially I found problem areas in the field of system development by interviewing practitioners and by consultancy. These initial problem areas were whittled down to: action modelling, entity model clustering and a diagrammer.

Action modelling is the modelling of detailed data behaviour using the same structuring concepts as data modelling. It was developed because of a lack of such analysis in systems development.

Entity model clustering is about aggregating the entity types in a large entity model to abstract the essential meaning and to identify the most fundamental entity types. It was developed because of a need to summarise large entity relationship models for usability and comprehension. It has been used widely and has many benefits.

A parallelism between data and activity modelling was developed as a result of the research into action modelling and entity model clustering. It needed the concepts derived from the other two areas to finally complete the theory, summarised as: every data modelling concept and structure has an exact equivalent in activity modelling and vice-versa. This theory gives a wholeness and completeness to modelling data and activity.

A diagrammer was produced for the automatic production and manipulation of entity relationship diagrams from a base description. These diagrams are the basic tool of the data modeller; automating them saves time and potentially raises their accuracy.

The main research problem was that few companies were willing to be guinea pigs, so most of the research was developed by thought 'games'. Most areas have been published in refereed publications as this was seen as the best way of establishing their academic credibility. All areas have been incorporated into or had an impact on James Martin Associates and their methodology Information Engineering, which provides a framework for coordinating the research areas.

This research can best be summarised as an attempt to find techniques for improving the systems analysis process.

CHAPTER A
INTRODUCTION

CHAPTER A INTRODUCTION

A1 SUMMARY OF THESIS

A1.1 THE RESEARCH

Information systems development typically consists of a number of 'stages' (a set of activities to be undertaken to develop a system), generally being **planning** for a range of systems, investigation of and **analysis** of each system, **design** of the systems and then **construction**. This thesis concentrates on improving the areas of planning and analysis. The results are presented for a particular methodology (set of tools and techniques) - **Information Engineering** [MACD,84], [MITC,85], [GIMA,85], but are applicable to any equivalent methdology, for example, SSADM, D2S2. Indeed, the research was originally carried out in the context of D2S2 but was later transferred.

Information Engineering has been under development for a number of years and has a rich past; its current form owes much to the work of Finklestein and Martin [MAFI,81], and to CACI Inc. International [MACP,82]. Information Engineering has been used successfully in some of the world's largest companies and is described in greater detail in appendix X1.

Before my research Information Engineering and the other

methodologies gave the appearance of being a 'hotch potch' of techniques due to their development. They also had certain deficiencies in the way they dealt with some areas. The main part of this thesis (part II) concentrates on two of these areas - coping with the modelling of complex and diverse areas, and the accurate modelling of the detailed logic of processes. These are both discussed in much greater detail throughout the thesis. The results achieve a secondary effect of consolidating the methodologies and giving them a more coherent appearance. This is discussed in Chapter D.

The research commenced in September 1983. It has been undertaken in a number of places, principally Thames Polytechnic, London and University of Warwick, Coventry, and with two industrial sponsors, CACI Inc. International and James Martin Associates. The results have been applied in some of Britain's largest companies (see sections B6 and C6), have been published in various places (see appendix X5), and have also been incorporated into the Information Engineering methodology.

The theme of the research is the development of methods and tools for the structuring of data and for modelling its behaviour, but is best summarised as 'how to improve the analysis process'.

I started by interviewing CACI's consultants to elicit their thoughts on problem areas in this context and distilled a number

of areas from this process. These areas were investigated in reasonable depth and three selected as being worth still deeper research: action modelling, entity model clustering and a diagrammer. See chapter F for a fuller treatment of the research process and for descriptions of the other areas.

In 1985 I changed sponsoring company from CACI to James Martin Associates (JMA), but by this time the ideas were well-formed. Fortunately the transfer was effected into Information Engineering with very few changes as the main techniques which the research was based on are common to both D2S2 and Information Engineering. These techniques are:

* entity relationship modelling
* process decomposition
* process dependency modelling
* process logic analysis.

A1.1.1 Action Modelling

When the research commenced, there was a noticeable gap in activity modelling when it came to modelling the detail (ie. the detailed behaviour of data). Action modelling was developed to fill this gap by utilising the structuring concepts of entity relationship modelling which had proven very successful. The resulting model, an activity relationship model (or action model), looks quite close to a dependency diagram, but is more complete and comprehensive than one.

Action modelling uses the concepts of dependency, optionality, conditionality, cardinality, exclusivity and abstraction exactly as they apply to entity type relationships, including the diagrammatic conventions. The activities modelled in this fashion are usually sub-elementary processes (known as actions) but almost always are above the basic action level (ESTABLISH, UPDATE, DELETE, SELECT, etc.). A novel concept which was introduced by action modelling is called 'floating' action. This has many uses and benefits, but primarily it gives a flexibility of representation. 'Floating' actions are just that, actions with non-specific dependencies; an action may be dependent on other actions but this is not specified explicitly, just implicitly by specifying the action's pre-conditions.

Action modelling has been used at International Paint and has been incorporated into Information Engineering.

A1.1.2 Entity Model Clustering

Entity relationship models are mainly used for communication purposes. However any large (>30 entity types) entity relationship model becomes difficult to draw and communicate; it is really only the producer of the diagram who understands the diagram fully. Entity model clustering was developed to deal with this problem and is concerned with structuring an entity relationship model by 'association clustering' to improve its communication and maintainability. It is described in chapter C.

These improvements come about by the controlled use of abstraction to extract the essential information at one level while leaving the total detail at a lower-level in small 'chunks' or clusters. The effect is akin to data flow diagram structuring applied to a data model. The result of clustering an entity relationship diagram is a 'tree' of entity relationship diagrams at different levels, with a 'box' on one diagram being decomposed into a lower-level diagram (except for the lowest-level of course).

The concepts of entity model clustering not usually involved in an entity relationship diagram are major entity types, subject areas and inter-diagram connectors. Major entity types are the most fundamental objects in an organisation, for example, such things as Customer, Supplier, Product and Organisation Unit. Subject areas are groupings of entity types according to some criteria which are normally functional in nature, but could be entity types all concerned with a major entity type, eg., a Customer Details subject area. Subject areas form intersections between major entity types and appear as a single diagram. Inter-diagram connectors are needed to cope with inter-subject area relationships.

Entity model clustering has been used in some of Britain's largest companies such as Whitbread and Co. plc (where it was developed), the Prudential Assurance Co., Northern Gas, Calor

Gas and Sedgwick Insurance Brokers Ltd. It is now an accepted part of Information Engineering.

A1.1.3 Parallelism Between Data and Activity Modelling

Action modelling and entity model clustering are the main thrusts of the research. However they provided the basis for a research offshoot, the development of a theory of data/activity symmetry. In the past people have taken many different views in this area. Initially approaches were purely activity oriented; any data was just there to support the processing and was not interesting in its own right. In response to this, approaches were developed that took totally the opposite tack, data was the be-all and end-all of everything, and activities were not interesting. Nowadays a compromise approach has been developed where data and activity are more-or-less given an equal standing; the problem was that the available techniques did not reflect this equality totally.

The introduction of entity model clustering and action modelling goes a long way to correcting this; hence the research offshoot. What we find is that every data modelling technique and concept has an equivalent in activity modelling and vice-versa. This is not to say that data and activity are the same; they are not, for instance, you cannot 'execute' basic facts (eg. Paul is a Person), though it is possible to consider activity as data. The theory is that the structure of data and activity can be

modelled in exactly the same fashion, and hence represented in the same format in a computer system. This is discussed in greater detail in chapter D.

A1.1.4 Diagramming Tool

Developers can spend many man-days, weeks or even months drawing and maintaining diagrams which are otherwise very useful for system development. The danger is that the diagrams will contain mistakes and will not be maintained properly.

A diagrammer was developed as part of the research to automatically 'draw' entity relationship diagrams from a data dictionary definition, so taking the pain out of producing diagrams. It also has a maintenance module to allow the upkeep of the diagrams. As action model diagrams use the same basic conventions as entity relationship diagrams, the same tool could be used for action modelling. The Diagrammer is discussed in chapter E.

The Diagrammer was developed largely as a research tool but, as described in chapter F, this was abandoned due to a lack of facilities. I have been informed that CACI Inc. International, for whom it was developed, have taken the tool and modified it into a commercial tool.

A1.1.5 Knowledge Engineering

Another aspect of the research was to investigate the applicability of the techniques to areas of computing other than just information systems, in particular knowledge engineering. What was found was that the knowledge acquisition process is essentially the same for all types of system. This is discussed under the appropriate chapters (B & C).

A1.1.6 Research Problems

The main problems faced in the research were due to the nature of the research area. For example, one problem was how to validate the research. The techniques were intended to be commercially applicable and an environment was needed to try them out for the first time. Entity model clustering was developed at Whitbread & Co. plc so was less of a problem than action modelling. A similar problem was a lack of material to experiment with; most companies are loathe to lend out commercially valuable information. Problems were also encountered in gaining consultants' acceptance of the ideas, mainly because they wanted to have proof of their effectiveness in a commercial situation first. These problems, and others, are discussed in chapter G.

A1.2 DESCRIPTION OF DOCUMENT

This thesis is structured into three parts, an overview (I) (this part), details of the actual research (II), and various bits and pieces needed to help explain and conclude the research (III). Parts, with roman numberals, breakdown into chapters (alphabetic), which break down into sections (with arabic numerals).

A1.2.1 Overview Part

This overview part contains all the scene-setting material for the research. It gives an overview of the research, the research framework and a summary of the research.

A1.2.2 Research Part

The research part describes the results of this research project

The part is structured into five chapters:

B ACTION MODELLING

C ENTITY MODEL CLUSTERING

D PARALLEL BETWEEN DATA AND ACTIVITY MODELLING

E DIAGRAMMER

B and C are the main research areas and ideally would be read in parallel! They are of equal importance and neither is dependent on the other.

D brings together the strands of research described in B and C and uses them to put forward a theory of parallelism between data and activity modelling.

E is a side issue which was researched early in the project and had good results.

A1.2.3 Conclusions Part

There are various supporting pieces of documentation needed to complete the thesis. These are contained in the Conclusions part. There are six 'chapters' in this part:

F RESEARCH METHOD

G RESEARCH PROBLEMS

H CONCLUSIONS OF RESEARCH

I ACKNOWLEDGEMENTS

J REFERENCES

X APPENDICES

F discusses particular research issues.

G discusses various problems faced in carrying out the research.

H summarises and concludes the research and briefly discusses its successes.

X contains various appendices which are needed to complement the research description.

A2 SYSTEMS DEVELOPMENT AND ITS ENVIRONMENT

A2.1 INTRODUCTION

This research project is mainly concerned with entity relationship modelling, which is used in the field of systems analysis, which in its turn is part of the wider field of systems development.

Systems development is concerned with the production of systems. Systems can be classified in a vast number of ways. The Oxford dictionary defines a system as:

> "1. Complex whole, set of connected things or parts, organised body of material or immaterial things. 2. Department of knowledge or belief considered as organised whole..."

Within the context of automation a system can take on a number of facets such as batch or real-time, operational or decision support, closed or open [SOMO, 81].

We need to be able to build these systems as efficiently and effectively as possible. The problems involved in achieving this are wrapped up in the management of personnel and the complexity of system produced; it is simpler to product a closed, batch, operational system than an open, real-time, decision support

system - though the latter often has greater benefit to an organisation than the former.

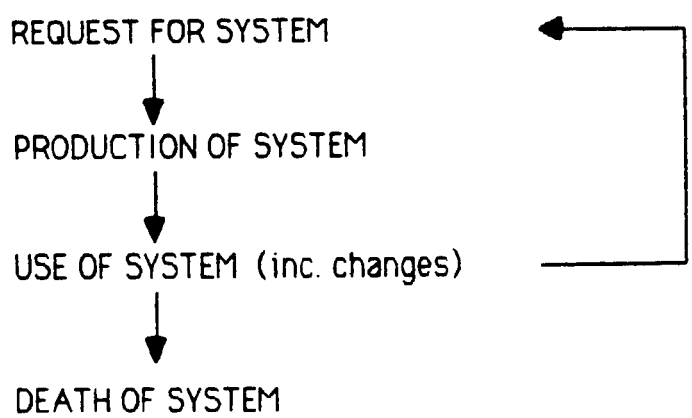
The discipline of efficient and effective production of systems is **system development**. Over the past ten to fifteen years a large number of methodologies to control the development process have themselves been developed, for example, Information Engineering [MACD 84], [MITC, 85], [GIMA, 85], D2S2 [ROEV, 81], [MACP, 82], ACM/PCM [BRSI, 82], ISAC [LUND, 79a], [LUND, 79b], [LUND,82], JSD [JACK, 83], [WILS,85], and many more. I will discuss these in further detail later on.

Each of these has an underlying **framework** which is the basis for controlling development. The vast majority of these frameworks are based on a 'top-down' approach in that they proceed from the general to the particular. The benefit of working in such a manner was recognised for software engineering as a part of system development long before it was applied to the whole of system development [DIJK,72], [WIRTH, 71]. (For system development, I have only found one methodology which does not profess to be top-down and that is JSD [JACK, 83] [WILS, 85].)

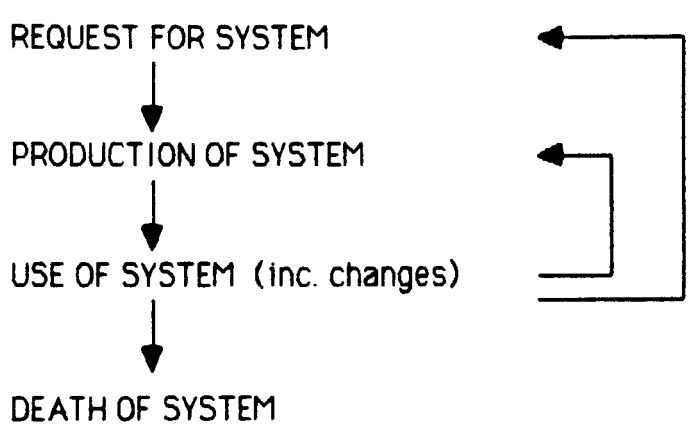
A2.2 SYSTEM LIFE CYCLE

All of the methodologies that I have looked at are based on the life-cycle concept, some more strongly than others. The

life-cycle concept considers the whole of the 'life' of a system from its inception through usage and on to its death, ie. when it is no longer used [ZAHN, 83]. Ideally, the death of a system coincides with its replacement by a new system in the 'use' part of the new system's life. Basically, the life-cycle is :



There are variations on this, for example, the evolutionary approach [RTW, 82] where a complete system is built and used bit by bit, for example :



But we are talking about basically the same thing.

A2.3 FRAMEWORKS

As I have already said, we are concerned here with the production of systems through systems development. In keeping with the life-cycle, a number of 'frameworks' have been produced that enable the management of development, usually as part of a methodology, but not necessarily. A framework is a proposed 'splitting' of the development process into manageable 'chunks'. Examples of some frameworks are contained in [ISAD, 84], [MACD, 84], [BRIT, 80], [LUND, 82], [CCA, 78], [ESA, 82], to name but a few.

One of the earliest frameworks was that produced by the NCC [LEE, 78]. This is often known as the 'Traditional' approach [WHFI, 82]. The chunks identified by Lee were:

Feasibility Study
System Investigation
Systems Analysis
Systems Design
Implementation
Review and Maintenance.

Review and maintenance correspond to the 'use of system' part of the life-cycle.

Feasibility study covers the investigation of a request for a system to evaluate if it is worthwhile; Systems Analysis is the investigation of a systems requirements in depth; Systems Design is the design of a system based on the requirements; and Implementation is the actual programming and data structure production of a system.

The traditional approach represented a breakthrough at the time because it provided a framework for management and control. However it suffered from a number of problems, mainly a focussing on a single application system to the detriment of a set of cooperating systems (allows no overall planning or architecture), a lack of opportunity for consultation by the users of system, an excess of tedious computer-oriented documentation and a too-rigid framework causing too-rigid control [AVFI, 86], [WHFI, 82], [SOMO, 81]. In addition this approach concentrated on the functional aspects of a system with no serious consideration of data.

These problems stimulated various different strands to overcome them. Some of these developments are the 'participative' approach (eg. [MLH, 78]), the use of 'prototyping' [BBW, 77], [DEMA, 83], the use of planning approaches [LUND, 82], structured analysis and design methods [ROSS, 77], [deMA, 78], [GASA, 79], and the database approach [PROW, 80], [SHAVE, 81], [MACP, 82].

A recent publication "Information Systems Development: a Flexible Framework" [ISAD, 84], of which I was one author, has tried to bring all these strands together, encompassing all the developments mentioned. The ISADWP framework is:

Business Strategic Planning
Information Systems Strategic Planning
Information Systems Tactical Planning
Analysis
User Design
Technical Design
Construction.

There are opportunities to use feasibility studies to check the viability of a system at any point during its development.

This framework is flexible, allowing the use of any approach, and attempts to maximise the involvement of users. In this research I have used the framework that underlies Information Engineering [MACD, 84], [MITC, 85], which is discussed in appendix X1.

The actual sequence of stages or phases depends largely on the approach taken and the development technology used. Most opportunity for flexibility is in the analysis stage and its interface with design.

A2.4 METHODOLOGIES

As mentioned, a single technique will cause problems. Modern methodologies have overcome this by combining a whole set of techniques in each stage of development. Indeed the LBMS methodology [HALL, 82], [BURC, 85], has been described as a 'cookbook' approach to development because it contains so many techniques.

Recently there have been a number of studies of methodologies, most notably the work of the IFIP TC8 WG 8.1 in its set of CRIS conferences [OLLE, 82], [OLLE, 83] which have tried to compare and contrast a number of methodologies.

The methodologies considered in the CRIS1 conference were: SYSDOC, ACM/PCM, CIM, SDLA, ISAC, D2S2, DADES, IML, Remora, EDM, ISSM, NIAM and USE. I do not propose to describe each of these here; interested readers are directed to the conference proceedings [OLLE, 82].

Another forum for discussion of methodologies has been the British Computer Society's Database Specialist Group in a set of two conferences "Data Analysis Update" in 1982 and "Data Analysis in Practice" in 1985. Again interested readers are directed to the proceedings [BAKER, 82] and [HOLL, 85] respectively. The first conference considered ICL's method, LSDM

(from LBMS and chosen under the acronym SSADM (Structured Systems Analysis and Design) by the UK Government's agency, the CCTA, for civil service developments)), an IBM method, BIS's method and Information Engineering, among others. The second conference considered Information Engineering, D2S2, a method from Whitbread (including part of this research project), JSD and EXTIM.

These forums have really been for a discussion of the detail of each methodology. CRIS2 [OLLE, 84] attempted to compare some of the methodologies, and papers loosely deriving from this conference are still appearing (eg. [FLYNN, 84], [FSW, 85] and [MADD, 83]).

As stated earlier, all the methodologies are based on the life-cycle, and most more or less follow a standard framework of Investigate-Design-Construct. Tozer [TOZER, 85] has shown how most of the methodologies only concentrate on a part of the framework. There are few which consider all of Planning-Analysis-Design-Construction.

Information Engineering is one methodology which does, though it is weak in the Construction area. ISAC [LUND, 79a], [LUND, 79b], [LUND, 82] is mainly concerned with Planning and Analysis. Most of the other CRIS methodologies are concerned with Analysis and Design.

A2.5 ANALYSIS APPROACHES

Fitzgerald and Wood-Harper produced a taxonomic study of analysis approaches [WHFI, 82]. They classified the approaches and their underlying paradigms into:

General Systems Theory
Human Activity Systems
Participative
Traditional
Data Analysis
Structural Systems Analysis

The General Systems Theory approach, based on the General Systems Theory [VonB, 68] is "an attempt to come to terms with and understand the nature of systems" [WHFI, 82]. Its problem is that is is too general, hence not easily applicable. We will not consider it further as there are no systems development methodologies based on it.

The Human Activity Systems Approach is largely based in the work of Checkland [CHEC, 81]. It is ostensibly a derivation of the General Systems Theory approach applied to the solution of 'soft' problems, ie. those which are ill-defined in nature. "It generates understanding of the environment and leads to possible structural, procedural, attitudinal or environmental change"

[WHFI, 82]. This approach is really concerned with defining the context of a problem rather than defining solutions.

The Participative approach is concerned with involving the users of a system in its design, ideally to the extent of the users designing the system themselves with the aid of a technical 'facilitator' where necessary [MLH, 78], [MUMF, 85]. This approach borrows from the work of the Tavistock Institute and their socio-technical methods for improving work systems, of which automated systems are one aspect nowadays. As Herscheim points out [HIRS, 83], the participative approach is considered very good in a work situation, but there are many practical and political problems with such a heavy involvement of users in system development.

The traditional approach has already been discussed.

The Data Analysis approach is one of the main bases of this research. It is based on the assumptions that data is the central aspect of any system and that it is more stable to change than the activities which make use of the data. It has largely been based on the work of Chen [CHEN, 76], Palmer [PALM, 78] and his work at CACI [SHAVE, 81], [DAVE, 80], [ROEV, 81], [MACP, 82], [ELLIS, 82], [ELLIS, 85]. Tozer [TOZER, 76] and Flavin [FLAV, 81] have also documented approaches. Due to the heavy concentration on data, the data analysis approach has tended to suffer from a lack of consideration of activity. The

approach also requires a reasonable amount of skill to be successful. Additionally a vast amount of documentation is often produced, needing some form of automation to cope with it [SOMO, 81], [MACP, 82]. This approach is the basis of Information Engineering (see appendix X1) and, hence, my research.

The Structured Systems Analysis Approach largely derives from the work of Yourden and Constantine [YOCO, 75], which was improved and made popular by de Marco [deMA, 78] and Gane and Sarson [GASA, 79]. The main basis of this approach is the use of Data Flow Diagrams, which is the main problem of this approach; reliance on a single technique will always cause problems as it only gives a single viewpoint.

These approaches are in essence complementary and aspects of them could all be used at the same time.

Most of the methodologies mentioned in A2.4 agree with aspects of all these approaches but none considers them all. ISAC, for example, makes use of the Human Activity approach, some of the Structural Systems Analysis approach and some of the participative approach, Information Engineering makes use of the Data Analysis, Participative and Structured Systems Analysis approaches. NIAM on the other hand is heavily imbued with the Data Analysis approach, while LSDM uses Structured Systems Analysis tainted with Data Analysis.

Researchers have attempted to take some of the methodologies and expand them into other areas or stages of the framework. For example, Iivari and Koskela [IIKO, 83] have produced new methods for ISAC to enable it to cope with greater amounts of detail.

The approach that a methodology takes and its intention tend to reflect its origins quite heavily. For example, Scandinavian methodologies such as ISAC, CIM and ISSM reflect the ground work of Langefors in the 1960's and his seminal work: "Theoretical Analysis of Information Systems" [LANG, 66]. They tend to be based on the 'infological' stance taken by Langefors. On the other hand Information Engineering, D2S2, NIAM and ACM/PCM (Active Component Modelling/Passive Component Modelling) reflect their origins in producing database-oriented systems and are heavily centred on data analysis.

A2.6 DATA MODELLING

Most methodologies have some form of data analysis. Where they differ is in the type of data model they use.

Tsichritzis and Lochovsky in their comprehensive study of data models [TSLO, 82] include three low-level types of data model - the relational, network and hierarchical models (for prime references see [CODD, 70], [CODA, 71],[BACH,69], and [IBM, 75]), which are really outside the scope of this study. They also

include four higher-level data models (conceptual models), the entity relationship, binary, semantic network, and infological.

The entity relationship model was first described by Chen [CHEN, 76]. Later references to this include [SHAVE, 81], [PARK, 82], [ROEV, 81], [DAVE, 80], [VERY, 84], [FLAV, 81] among others. There has also been a succession of conferences devoted to the technique [CHEN, 80], [CHEN, 83], [DJNY,83], [IEEE, 85], though other types of model have been discussed there. The basis of the entity relationship model is an identification of significant groups of data - entity types - and the relationships between them. This approach has had great success for various reasons and forms a basis of a number of methodologies, eg. Information Engineering and LSDM. It is also the main base of this research.

A binary model is "any graph data model in which the nodes represent simple, single attributes and the arcs represent binary relationship types between two attributes" [TSLO, 82]. Most binary models allow the building up of groups of nodes into higher-level concepts. The main forces behind this model are Abrial [ABRI, 74], Senko [SENKO, 75], Kent [KENT, 78],[KENT,83]. In fact an entity relationship model is a restricted form of binary model, with the attributes grouped into entity type nodes which are the only objects that can have relationships defined. NIAM [VEBE,82] has a binary model where entity types and attributes (NOLOTS and LOTS - non-lexical objects types and lexical object types) are identified and related.

Semantic network models originate from artificial intelligence work. They are distinguished from previous models because "the goal of these networks is the representation and organisation of general knowledge of the world as opposed to specific business applications" [TSLO, 82]. There is little inherently different about the semantic network model and the previous models, the difference arises through use and intention. Semantic network models have been applied to non-artificial intelligence areas, most notably by Hammer and McLeod [HAMC,78]. Brodie [BRSI,82] has made use of this to produce ACM/PCM.

The infological model is intended to provide as natural a model as possible for communicating with people. This is intended to be used for capturing requirements, which are then translated into computer representations - datalogical models. The distinction between infological and datalogical was first made by Langefors [LANG, 63], [LANG, 69]. As discussed earlier, the Scandinavians have made much of this, particularly Bubenko [BUBE, 80]. A notable British use of the concept has been Stamper with the LEGOL project based at the London School of Economics [STAM, 77]. The LEGOL project is an attempt to apply formalisms and strict rules to the analysis process; it applies less formalisms to the infological realm than the datalogical realm due to the different uses - any restrictions placed on an infological representation reduce the naturalness of the representation.

The entity relationship model as applied in Information Engineering and D2S2 is intended to be infological by providing a method of representing every day language [ELLIS, 82]. The main differences between this and the accepted infological models is the degree of formalisms applied and the use of object derivation; entity relationship models are ideally non-redundant, whereas redundancy forms a large part of people's everyday life.

A different type of data model is a 'conceptual graph' [SOWA, 84]. These are only used in artificial intelligence and will not concern us here.

Another consideration is the abstraction of data, abstraction being the process of summarising a set of objects into some higher-level object. Smith & Smith are the main parents of this [SMIT, 77a], [SMIT, 77b] and identified two forms of abstraction: aggregation and generalisation. Brodie [BROD,83] expanded these giving association and classification. These concepts are considered in appendix X1.

Many other people have built on these ideas. For example, Lee and Gerritsen have investigated generalisation further [LEGE, 78]; Bolour & Dekeyser have considered the concepts of abstraction as applied to time, finding four types of abstraction here: time, identity, circumstance and

absolute/relative abstraction [BODE, 83]. Vermeer [VERM, 83] describes various concepts and heuristics for forming abstractions in a conceptual schema.

A2.7 ACTIVITY MODELLING

Much less emphasis has been placed on modelling activities than data, primarily because there is no equivalent of a database, so there is much less need to provide exactly the right activity structure.

The most common method of activity modelling is data flow diagrams as discussed previously. There are other methods, most of them actually based on interacting with data.

A good example of one of these is Rosenquist's work [ROSE, 82] concerning the analysis of activities that affect an entity type causing it to change state. I am not sure of what caused what, but this technique appears in various forms in LSDM [HALL, 82], [BURC,85], Information Engineering, [MACD, 82],[MACD,84], D2S2 [MACP, 82], and JSD [JACK, 83], [WILS,85], among others.

Information Engineering and D2S2 also have a form of activity modelling called 'dependency modelling'. This is basically a restricted form of data flow diagram, being more or less the same thing but without 'data stores'.

Hamilton & Zeldin have done some very good work in activity modelling with their HOS 'methodology' [HAZE, 75], [HAZE, 79]. They proved that by the use of three basic primitives - sequence, selection and parallelism - the decomposition of an activity (ideally into a program) could be proved to be correct. Of course this still leaves the problem of ensuring that the correct requirement has been specified to be decomposed from.

A number of methods are based on using language as a specification tool. A classic example of this is SYSTEMATICS [GRIN, 66], one aim of which is to produce "a statement of requirements which is complete, unambiguous, short and free from programming strategy" [GRIN, 75]. It is actually designed as a data specification and activity specification language. The activity specification is well explained in [GRIN, 79]. Sernadas has discussed its use as a query language in its own right [SERN, 81a], [SERN, 81b].

PSL/PSA is another example of a language used for system specification [TEHE, 77]. PSL stands for 'Problem Statement Language', being a language for describing the results of some analysis as a problem, and PSA - 'Problem Statement Analyser' - is an automated analysis of this. PSL/PSA has been widely used as a basis for further research, eg. [BOPI, 79],[TMHY, 80].

There has also been a lot of work at the USC Information Sciences Institute, eg. [GOWI, 80], [BALZ, 80], [BAGO, 79],

[BALZ, 81], based around producing language functional specifications among other things.

Some language-based functional specifications described in [RIAL, 78], [LIND, 79], [RIDL, 79], are concerned with identifying events and using these as the basis for the specification, ie. the specification is event-based, with events used as the main structuring concept.

Another approach to achieving good activity requirement specification is to use a predicate-based language, such as Prolog [KOWA, 79], [CLME, 81]. Some methodologies rely on this method.

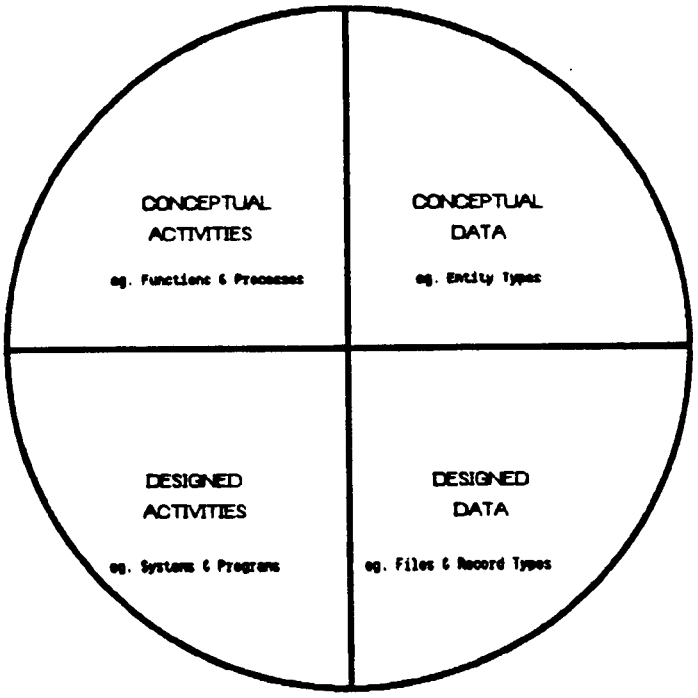
Problems arise from expecting users to comprehend any language-based specification. Personally I doubt that any language specification is usable as the ideal specification must be comprehensible by users for communications and verification purposes if nothing else; structured text is rarely easily comprehensible.

Remora [RORI, 82] have a diagrammatic method of describing certain event concepts called 'Direct Systematic Chronologic Dependency' and 'Direct Conditional Chronological Dependency' which appear to be more-or-less a dependency diagram of the life cycle of an entity type.

A2.8 AUTOMATION OF DEVELOPMENT

Development methodologies are concerned with providing the best mix of automated and manual aspects in a work-system. However it is only recently that similar efforts have been put into the development process itself.

A classic example of this being done is the work of the BCS Data Dictionary Systems Working Party (DDSWP) with its classic report [DDSWP, 74] and its later Journal of Development [DDSWP, 82]. One of the lasting impacts of this work is the concept of meta-data, being a description of a data object. For example, an entity type has properties of name, description and attributes among others, which apply to all entity types; these properties are meta-data about entity types. Perhaps a more important impact has come from their quadrant diagram:



(adapted from [DDSWP, 74]).

This diagram has been used and modified many times, but is rarely questioned, if ever.

Data dictionaries have become an accepted part of everyday system development. Good descriptions of many commercially available data dictionaries are contained in [BAKER, 83].

Wilson [WILS, 81] describes the results of the MU5 project at the University of Manchester, which considers the automatic production of design documentation and code production.

Another aspect of automation is providing graphics support for the diagrams involved in analysis. SYSTEMATOR [ASMO, 82] has a crude diagramming tool, and ISAC has had some experiments in this area [LUND, 83]. Various products are now on the market, for example, EXCELERATOR is a collection of diagramming tools with an underlying data dictionary. The Information Engineering Facility (IEF) being produced by James Martin Associates and Texas Instruments is a fairly sophisticated set of diagramming tools and system encyclopaedia. Some notable work has been done in this area in Italy, especially at the University of Rome [TBT, 83]. This has resulted in a product called GIOTTO [TAMA, 85], [BFN, 85]. Their work has relied on the application of graph theory to the automatic production of diagrams. Chan and Lochovsky in [CHLO, 80] describe yet another diagramming tool.

It is not sufficient just to record information captured through such diagrammatic tools, it must be consistent, especially if it provides the basis for automatic generation of systems. The IEF and its failed predecessor, The System Factory from CACI Inc. International [CACI, 83], were aimed at this. PSL/PSA [TEHE, 77] was one of the first attempts in this area and proved successful. [TMHY, 80] described an extension to it. The trend that this exhibits is towards a complete integrated tool (IEF is intended to be this). INCOD as described in [ABLV, 83],[ATCA, 83] and [ABCDLVZ, 83] is an Italian attempt at this. INCOD stands for Interactive Conceptual Design of Databases; it uses an entity relationship model to describe the static aspects of an system but uses a specification language for the dynamic aspects.

All of this Italian activity is part of an initiative called DATAID. DATAID consists of a methodology and automation for database design. The initiative and some of the results are described well in [CERI, 83].

A recent Butler-Cox report discussed many of the issues of system building tools [BC, 85]. The tools it considers are not as advanced as many of the ones discussed above (being basically fourth generation languages) but the issues are the same.

A2.9 OTHER ASPECTS OF SYSTEM DEVELOPMENT

A2.9.1 Good Design Principles

What makes a good method for system development? Somogyi [SOMO, 81] praises ISAC for simple documentation as "... it uses

- A minimum amount of text
- As little formality as possible
- Diagrammatical representation
- Very few symbols",

and for using the same analytical methods and documentation techniques throughout the complete development process. This is the design aim of Information Engineering as well [FMM,86], ie. to produce simple, usable, communicable representations and methods for system development.

Another important consideration is splitting a requirements specification from implementation considerations. This is so the basic requirement can be stated without worrying about the technology it is going to be implemented on. However, as Balzer and Swartout point out, this is not always possible as some requirement issues are inextricably linked with implementation [SWBA, 82].

As Longworth describes, some necessary requirements of a methodology for analysis and design are:

"- To obtain a detailed overview of the system with the intention of breaking development down into several simple, interrelated but separately implementable parts
- To achieve the fast development of each separate part of the system
- To use methods to prevent coding bugs and to find logic ... errors as early as possible in the (development) cycle
- To achieve effective control of development and change" [LONG, 82].

Gray's empirical research [GRAY, 84] pointed to a number of factors which are considered important for evaluating a system development methodology. These were: improved systems definitions, maintainability and ease of enhancement of systems, productivity in development, accurate prediction of time and costs required for development, control of time and costs during development, recruitment and retention of suitable staff, user involvement, planned development of staff skills and control of staff resources.

In practice methodologies are not seen as the be-all and end-all. In a Xephon survey structured methods "were reckoned to be the least effective of seven techniques for reducing the

applications backlog". In order these techniques were ranked:

"1. On-line programming
2. More computing resources dedicated to development
3. Application generators
4. More development Staff
5. End-user computing
6. Application packages
7. Structured methods"

[SW, 84].

It is possible to undertake exhaustive analyses of methodologies. CRIS2 was a conference devoted to this subject [OLLE, 83]. Iivari and Kerola put forward a framework at that conference for such an analysis in the form of a detailed questionnaire [IIKO, 83]. At the conference Wassermen, Freeman and Porcella also described a questionnaire and its results [WFP, 83].

A2.9.2 Prototyping

Prototyping is being proposed and used more and more nowadays for system development. Prototyping in system development is basically the trying out of a system before its formal use so it can be improved and re-developed where necessary.

There are two types of prototyping which can be used, the

display of a series of screens to a user to validate the basic system structure, and a full-blown 'mini'-system to validate all aspects of the system.

The second type is proposed by Bally et al [BBW, 77]. The effect of prototyping on the systems development process is discussed by Britten in [BRIT, 80]. Basically, he proposes a specify-build-evaluate loop until a satisfactory system is built. Dearnley and Mayhew discuss this issue further and give guidelines for its effect on the life cycle and say that "it forces the analyst to consider the benefits of building a prototype" [DEMA, 83]. Prototyping has been used to attempt to justify the invalidity of the life-cycle [McJA , 82]; but the result is just a more flexible life-cycle.

A2.9.3 Quality of Information Systems

The aim of system development methodologies is to produce high-quality information systems. The question then arises of how to evalute and ensure the quality. There has been research in this area, but this is on the periphery of this thesis. Orman [ORMAN, 83] discusses an evaluation of information systems. He characterises them and their desirable features. He proposes 'Information Independence' as a "useful and quantifiable measure of the value of an Information System".

When developing models their correctness is often left to

chance. Information Engineering has a task to check this for its models. Lundberg [LUNDB,83] describes criteria for the correctness, consistency, satisfiability and completeness of information and the relationship between information models and conceptual schemas. Bubenko [BUBE, 77b] gives a similar discussion.

Not only does the development process have to be of good quality, but also the data in a system. It does not matter how good the basic system is, if its data is of poor quality then the system is of little use. Reductions of quality can happen for a number of reasons, primarily "delays in processing times, lengthy correction times and overly or insufficiently stringent data edits" [MOREY, 82]. Morey discusses this and how to improve the quality.

Brodie [BROD, 80] discusses many of these issues and how to use the development process to improve the data quality of information systems.

A2.9.4 Time

The question of the representation of time is a thorny one. There are many posssible ways of dealing with it, but the most suitable seems to be to treat it as another informational element. A survey of the role of time in information processing is [BADW, 82]. In CRIS2 Kung analysed three methodologies for a

time perspective and gave some suggestions and improvements [KUNG, 83].

A common method of dealing with time is to extend a data model to cope with it. One example for entity relationship models is [KLOP, 83]. He describes TERM, a specification language with various extensions, eg. entity states and time data types.

Bubenko [BUBE, 77a] considers the effect of time on information modelling. He identified two types of time: extrinsic (being "the time when a particular assertion is made or conclusion is drawn") and intrinsic (an actual part of an assertion conclusion).

LEGOL is a project which attempts to formalise specifications. The time dimension forms an important aspect of this [JOMA, 80]. It is basically achieved through considering the time of establishment and termination of an informational element. A similar approach was taken by Ferg who proposes the setting up of time attributes and the definition of relationships between them and the entity types they apply to [FERG, 85].

Other considerations of time can be found in [LANG, 66], [ABRI, 74] and [SUND, 74], where again time is included as an informational element.

As described in A2.6, Bolour and Dekeyser used time as a basis for abstraction [BODE, 83].

A2.9.5 Decision Support Systems and Artificial Intelligence

There is currently great interest in the separate but linked fields of Decision Support Systems and Artificial Intelligence. The main types of systems we have been discussing up to now have been on-line or batch 'information' systems providing basic, structured operational or planning and analysis support to an organisation. Decision Support Systems tend to be strategic systems with an unstructured use.

Six types of decision support system have been identified [ISA, 84]:

Chief Executive Information Systems

Commercial Operational Analysis and Planning Systems

Industrial Operational Analysis and Planning Systems

Preference Determination Systems

Cognitive Mapping Systems

Expert Advisory Systems

As can be seen, 'expert systems' have been included in this list, because they are exactly that - expert decision supporting tools. Expert systems are systems which mimic experts. The most famous is probably MYCIN [SHOR,76]. They were described well by Michie [MICH, 80] and have since started to have great commercial success as well as academic rspectability. MYCIN itself has been made into a 'shell' [VanM,79], ie. the basic

inferencing mechanisms separated from the 'knowledge base' of expertise. MYCIN has even been 'rationally reconstructed' [CEBR,84]!

Another field of current artificial intelligence interest is the production of data flow hardware; hardware which carries out highly parallel processing where ideally a minimum of unnecessary serial processing is achieved [MALIK,83]. Such developments plus knowledge base systems form the backbone of the thrust into fifth generation computing [ALVEY, 85].

A2.10 THE RESEARCH IN PERSPECTIVE

This thesis is mainly concerned with the planning and analysis stages of the systems life cycle. It is presented in the context and framework of the Information Engineering methodology, but was initially developed for the D2S2 methodology. As a result the research is based in the data analysis, participative and structured systems analysis approaches. Current work suggests that it is equally applicable to other development methodologies that are also based on these approaches.

Within this environment, the research investigates data analysis and activity analysis based on the 'entity relationship' data model, and considers automation aspects where appropriate.

Investigation was also done to see if the research was applicable to more than just simple information systems by considering its applicability to knowledge engineering.

A THEORETICAL AND PRACTICAL INVESTIGATION OF TOOLS AND TECHNIQUES FOR THE STRUCTURING OF DATA AND FOR MODELLING ITS BEHAVIOUR

Volume II of III

by PAUL BARRIE FELDMAN B.Sc(Hons)

a PhD thesis

UNIVERSITY OF WARWICK
COVENTRY, ENGLAND

SCHOOL OF INDUSTRIAL AND BUSINESS STUDIES

September 1986

C O N T E N T S

LIST OF ILLUSTRATIONS

Illustration | Page

Chapter E

PART II

THE RESEARCH

CHAPTER B ACTION MODELLING

B1 INTRODUCTION TO THIS CHAPTER

This chapter discusses the technique of **action modelling**, which is one of the main parts of this research project. The other main part is entity model clustering, which is discussed in chapter C. These strands are brought together in chapter D.

In this chapter I first discuss why action modelling is needed (B2), ie. the reasons why action modelling was developed. I then consider the concepts which go to make up action modelling and how to actually analyse activities using it (B3 and B4). Some examples of action models are given in B5, followed by details of its use in a commercial situation (B6). Finally, B9 gives details of further research areas that have been identified.

Action modelling is a technique for the analysis of the inherent logic of processes. As such it firmly belongs in the Business Area Analysis stage of Information Engineering. There are really no new concepts introduced by action modelling; its originality lies in the combination of concepts and intended use. As the comparison (B8) shows, there is no other technique which analyses

CHAPTER B ACTION MODELLING

B1 INTRODUCTION TO THIS CHAPTER

This chapter discusses the technique of **action modelling**, which is one of the main parts of this research project. The other main part is entity model clustering, which is discussed in chapter C. These strands are brought together in chapter D.

In this chapter I first discuss why action modelling is needed (B2), ie. the reasons why action modelling was developed. I then consider the concepts which go to make up action modelling and how to actually analyse activities using it (B3 and B4). Some examples of action models are given in B5, followed by details of its use in a commercial situation (B6). B7 gives a number of benefits of using action modelling and B8 compares it to other techniques. Finally, B9 gives details of further research areas that have been identified.

Action modelling is a technique for the analysis of the inherent logic of processes. As such it firmly belongs in the Business Area Analysis stage of Information Engineering. There are really no new concepts introduced by action modelling; its originality lies in the combination of concepts and intended use. As the comparison (B8) shows, there is no other technique which analyses

the inherent logic of processes in the same way as action modelling; the benefits described in B7 show why things should be modelled in this way (it also solves the problems identified in B2).

B2 WHY ACTION MODELLING?

B2.1 FUNCTIONAL INSTABILITY

A common theme in current development methodologies is that data is more stable than activities. It is beyond the scope of this thesis to argue the point in any depth. However it is certainly unarguable that the results of design have this relationship, ie. a database is more stable than programs, simply because the consequences of changing the structure of a database are much more far-reaching than changing the structure of programs, thus it is avoided wherever possible. This philosophy affects analysis because it emphasises the importance of correct data over correct activity.

Another reason for data being considered more stable than functions is that data tends to be shared throughout an organisation. This rarely applies to activities, which tend to be specific to a given part of an organisation. Therefore data is considered to be more isolated from an organisation's structure than activities and hence, more stable. Thus the stability argument becomes axiomatic to modern development methodologies.

The main counter-argument is that data must exist to support

activities, so data should be just as stable as the activities it is based on. There is a certain amount of validity in this argument except when shared data (data shared between activities) is considered. Here the data is partially isolated from the change of activity unless all the activities that share the data are affected. Thus data used in only one activity is as stable as that activity, but data shared between activities is more stable than any individual activity. (For the purposes of this discussion ad-hoc queries are considered to be activities; therefore data in on-line query systems is shared and, hence, is more stable than the activities, many of which have a very short lifespan.)

Typically activity modelling is much more complex than data modelling. Thus a lot of effort can be expended in a task whose results are thought to have a shorter lifespan than the simpler task of data modelling. It would be desirable if the results of activity modelling could have the same length of applicability as the associated data. This was an aim of action modelling. The results of action modelling have a close relationship with data which gives the results a much greater level of stability; lower-level actions should be as stable as the associated entity types.

A problem touched upon above is that data can be distributed throughout an organisation whereas the activities tend to be

specific to parts of the organisation. Therefore it is difficult to form organisation-free activity models with current techniques. This results in less stable activity models (not necessarily the actual behaviour) because they may need to be changed whenever an organisation's structure changes, a fairly frequent happening in the majority of organisations. The modelling of shared data usage, being the shared actions (actions shared between activities), should result in an elicitation of any functional parts which are shared throughout an organisation, because shared data usage should result from shared data. Thus the results of action modelling have a freedom from organisational considerations which cannot be achieved through current functional techniques.

B2.2 METHOD SIMPLICITY

Most methodologies have vastly different methods for capturing different information for systems development, eg. different methods for capturing data requirements than for capturing activity requirements. This causes problems in learning conventions and in applying the techniques involved. It also introduces an artificial separation between similar aspects of a system. For example, entity life cycles [ROSE, 82] and entity state diagrams [HALL, 82] represent the same information, but are considered to be different due to the diagrammatic conventions involved. Somogyi [SOMO, 81] praises ISAC for

employing the "same analytical methods and documentation techniques ... throughout the complete development process", as does Information Engineering. On the other hand D2S2 [ROEV, 81] employs totally different analytical methods and documentation.

Action modelling was an attempt to introduce the same methods and techniques into both data and activity modelling. With the move into Information Engineering (see chapter F) this became less important, however the results are as applicable to Information Engineering as they are to D2S2. Action modelling uses the same structuring concepts for modelling both data and activity, and the same diagrammatic conventions as well. This should reduce the amount of learning involved. It also has the side-effect of the 'theory' discussed in chapter D, of a parallelism of representation of data and activity.

B2.3 LEVELS OF ACTIVITIES

There are three levels of activity we are interested in: a general description of the operation of an organisation, **functions**, 'broad brush' usage to achieve a given aim, **processes**, and the detailed 'behaviour' of entity types, known here as **actions**. Functions show the general processing requirements of an organisation. Processes are the particular requirements to achieve an aim of an organisation and at a given

level correspond to database transactions as described by Gray [GRAY, 81]; this level of process is known as **elementary process**. A characteristic of an elementary process is that it leaves the organisation in a 'consistent' state on completion, as far as integrity and business policy rules are concerned.

The general description of functions and processes in an activity specification is required for a number of reasons. Firstly, so that the business activities that need to be supported are described. Secondly, so that these activities can be agreed upon by the user. Thirdly, so that the general applications are defined and agreed and fourthly, to enable the entity relationship model to be refined and validated.

The benefit of using processes is that they provide cohesive units of processing whose meanings are easily grasped by all concerned. However they do not provide enough information for the design of software. An important aim of analysis is the gathering of enough information to enable the 'correct' design and construction of software and databases to proceed smoothly [ISAD, 84] with a minimum of unnecessary interaction with the users. On completion of analysis, users should not need to be approached by designers for information which should have been captured by analysts and, more importantly, designers should have enough information to design without having to make assumptions which often turn out to be wrong (though this is not to say that a re-analysis of an area could not take place if it

were to prove necessary). These requirements are rarely addressed by existing methods which are either too superficial for gathering detailed functional information (for instance data flow diagrams, eg. [deMA, 78] or, contrarily, are too detailed and consider design information rather than requirements (for instance access path diagrams, eg. [ROEV,81].

This was a driving force in the development of action modelling. Elementary processes need to be broken down into components of processing which reflect the actual behaviour of data, the 'actions'.

B2.4 DEVELOPMENT EMPHASIS SHIFT TO ANALYSIS

The common view of Business Area Analysis is that it is mainly concerned with the collection of enough information for a system designer to be able to produce a system matching the requirements of the ultimate benefactors of that system. In the past this has been achieved in various ways and through a number of methods, for example Structured Systems Analysis [deMA, 78] and Entity and Function Modelling [ROEV,81], [MADD,78].

Modern developments have shown that this simple approach is not sufficient to meet the needs of the present, let alone the future. Some of these developments are: the rise in use of relational database [CODD, 70], the use of automated system

development aids, for example, the Information Engineering Facility and prototyping [DEMA, 83], and the rise in knowledge-based computing [MICH, 80]. The result of developments such as these has been to shift systems development emphasis from design to analysis due to the extra flexibility they introduce in the development process.

Relational databases have a more flexible structure than traditional databases [CODD, 70]; this reduces the amount of work needed in design, but increases the importance of ensuring that the data is integral and properly understood. Thus analysis is emphasised.

With automated aids the source of information from which systems are automatically produced is emphasised, again this is the analysis process. With these aids many parts of the design process have vastly reduced importance, or even become redundant. Also, due to the speed with which systems can be produced using these aids, if the design is unsatisfactory, then it is a simple matter to correct a design if it proves unsatisfactory. Any designer and, hence, any design aids should just need information provided from analysis. However analysis collects most of its information from people, many of whom are very busy and resent being interrupted. Therefore with development aids design is of decreased importance and analysis is of increased importance due to the much greater amount of costly human-human interaction.

The same is true of knowledge-base systems where the resultant system is of little use without the initial knowledge. Thus knowledge acquisition is of immense importance to these systems [JOHN,84]. However the design of knowledge-base systems is becoming a matter of course as the process becomes better understood, but also due to the use of 'shell' systems such as EMYCIN [VanM,79]. These provide a large part of the basic system at a stroke; the basic system is then directly elaborated by knowledge gathered during the equivalent of the analysis process with little need for design intervention.

The consistent themes through the above discussion are the increase in importance of communication with users and the decrease of importance of the design process. Most modern system development methodologies have developed very good methods of collecting analysis information from users for data and high-level activities (eg. functions and processes). However, as discussed in B2.3, a large number of these system development methodologies display a degeneration into design information collection when it comes to dealing with detailed functional information (ie. detailed data behaviour) (see comparison - B8). Those methodologies that do not degenerate tend to model the detail of behaviour in an incomplete manner (again see B8), for instance, just for single entity types.

Thus there is a need for a comprehensive method of aiding the

collection and communication of detailed functional information in a user-oriented manner at the analysis level. Action modelling is such a method. It was developed to use the same conventions as entity relationship modelling, thus aiding the communication and collection as only one set of conventions need be learnt by both users and analysts (see B7). It is comprehensive; I have not found any behavioural information which cannot be collected using it. Additionally it has been developed specifically to be user-oriented and not to collect design information.

B3 ACTION MODELLING CONCEPTS

Action modelling is concerned with modelling the behavioural requirements of data in the same manner as a data model. As this thesis is concerned with entity relationship models, action modelling is explained with reference to them.

Figure B3.1 illustrates the action modelling conventions used in this thesis and compares them to the conventions of entity relationship modelling.

An action model consists of an **action dependency diagram** and definitions of the actions. For some actions a more detailed definition in some other technique may be produced. This distinction between a model and a diagram is discussed in more depth in chapter C, but forms one of the bases of Information Engineering. A diagram is just a representation of a model, which itself is a representation of the business area under analysis.

B3.1 ACTIONS

An action is a specification of a particular use of an entity type that can range from the simple to the complex. An **elementary action** is that part of a sequence of processing

which is undertaken on a single entity type without involving any other entity types; ie. that self-contained processing undertaken on an entity type as part of the achievement of some ultimate purpose. Elementary actions may contain smaller actions (see B3.7) and can be grouped into larger actions (see B3.5). Often the occurrence of an action will change the state of an organisation, though this is not necessary; however an organisation's state can only be changed by the occurrence of an action. Actions in action modelling have been given the same significance as entity types in entity relationship modelling.

The concept of actions needs to be specified to have a detailed enough description of the required manipulations of data upon which to base later design tasks concerning both data structure design (to enable efficient usage) and for program design (to enable the content of programs to be designed).

ENTITY

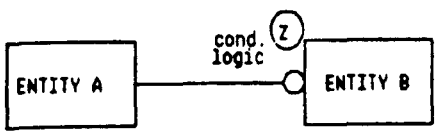
ENTITY RELATIONSHIP

CARDINALITY (1:N) -

many occurrences of B
are related to each occurrence of A

OPTIONALITY -

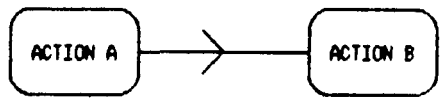
occurrences of B can exist without being related to occurrences of A, but not vice-versa



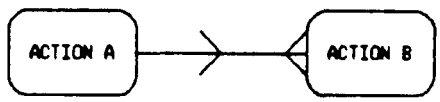
CONDITIONALITY -

the condition determining if occurrences of B are related to occurrences of A is defined by the condition logic, and referenced as condition (Z).

ACTION

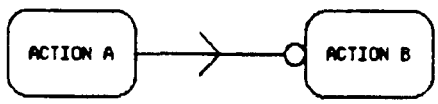


ACTION RELATIONSHIP



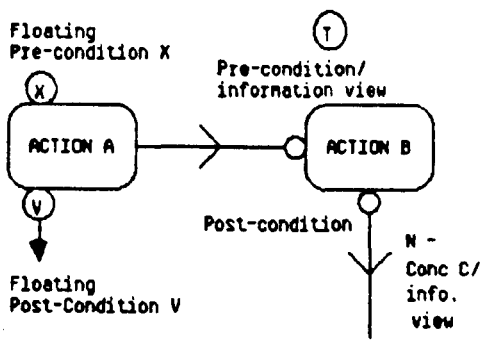
CARDINALITY (1:N) -

many occurrences of B happen for each occurrence of A



DEPENDENCY -

B can occur without A occurring, but not vice-versa



CONDITIONALITY -

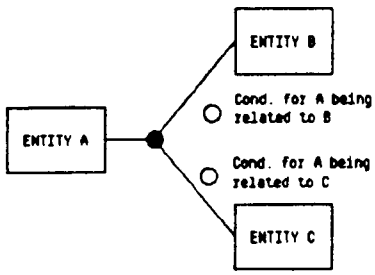
Pre- & Post- Conditions (both floating & not) and Conclusion C reached with probability N as a consequence of Action B

(See text for explanations)

Figure B3.1 Action Modelling Conventions and Entity Modelling Conventions (part I)

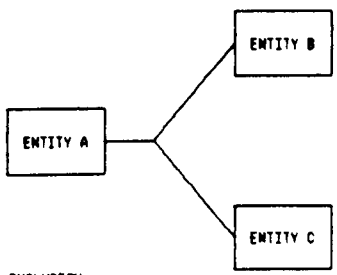
INVOLUTION -

an occurence of A can be related to other occurences of A



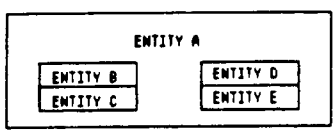
MUTUAL EXCLUSION -

at any one time an occurrence of A can only be related to either an occurrence of B or an occurrence of C, but not both



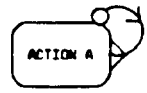
INCLUSION -

A has the same relationship to B & to C



ABSTRACTION (SUB-ENTITY TYPES) -

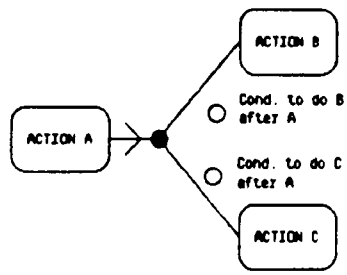
occurrences of B and C can be generalised to A, as can occurrences of D and E, so B & C aggregate with D & E to form A. An occurrence of A can either be an occurrence of B or an occurrence of C, and it can also be either an occurrence of D or an occurrence of E.



RECURSION -

occurrences of A may involve other occurrences of A - or -

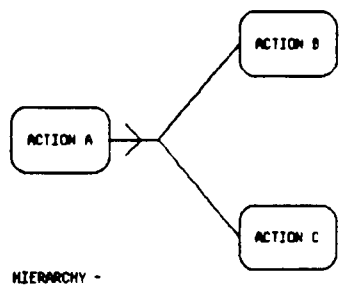
after an occurrence of A another A may occur



MUTUAL EXCLUSION -

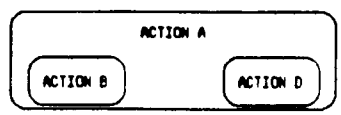
either B occurs after an occurrence of A or C occurs, depending on the conditions

(= IF Cond THEN B ELSE C)



HIERARCHY -

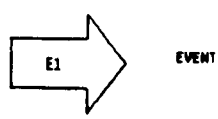
A consists of B & C in a hierarchical rel.



ABSTRACTION (SUB-ACTIONS) -

occurrences of B & C aggregate to form occurrences of A.
A consists of B & C in a hierarchical rel.

The . . . shows that A is decomposed, but the decomposition is documented separately



Documentary depiction of a triggering event (see text)

Figure B3.1 Action Modelling Conventions and Entity Modelling Conventions (part II)

B3.2 ASSOCIATIONS BETWEEN ACTIONS

Associations between actions express the dependency of one action on another; action associations are best described as 'one action changes the state of an enterprise so that the other action can occur'. For example, figure B3.2 gives an example of an action dependency; here removing the pen top changes the state of the 'universe' so that writing can occur, but there can be no writing until this is done.

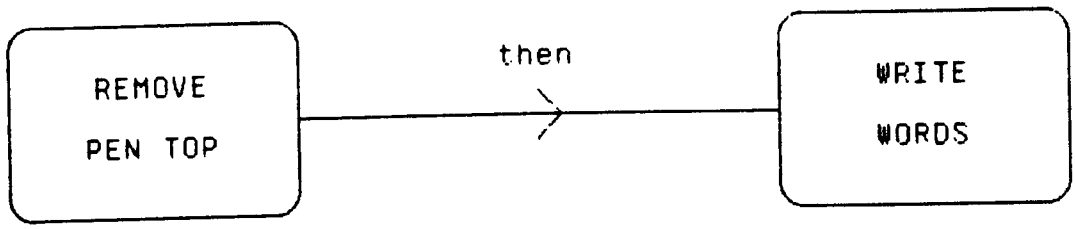


Figure B3.2 Action Dependency Example

B3.2.1 Optionality

Entity relationships also express dependency but with an additional concept of potential, though not necessary association; eg. in figure B3.3a a related occurrence of entity type A must exist for an occurrence of entity type B to be able to exist, but does not have to exist for an occurrence of entity type C to exist. The same concepts can be applied to behavioural specification, eg. in figure B3.3b action A must occur before

action B can occur but action C can occur even if action A does not; however, if action A does occur then action C must also occur. This concept of potential though not necessary occurrence is known as optionality.

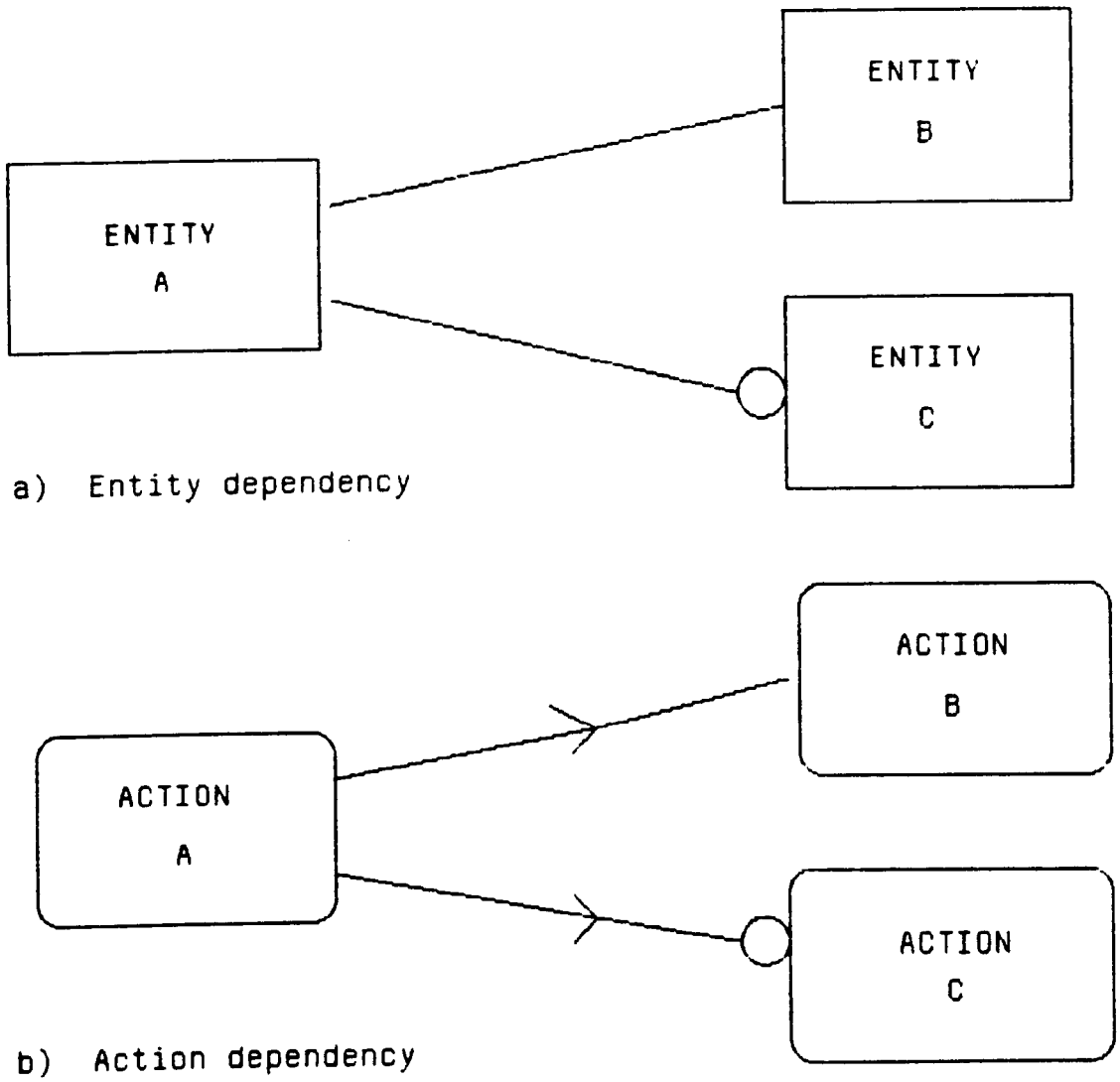
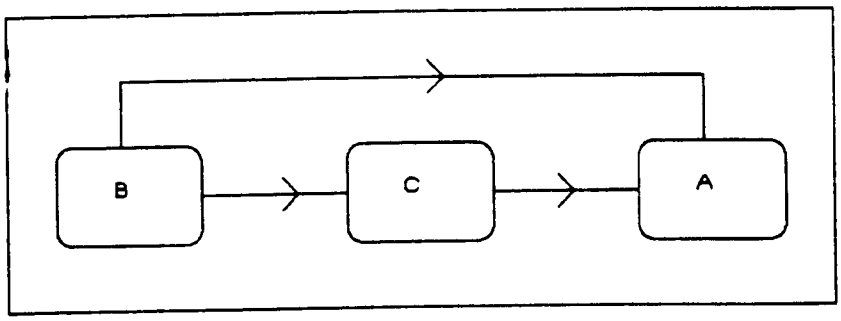


Figure B3.3 Action Relationships, Entity Relationships and Optionality

Action dependencies are a combination of time dependency and data dependency; the former because one action occurs before a related action, the latter because the first action changes the state of an enterprise's data so that it is in a sufficient state for the second action to occur. If an action can occur in given circumstances without this sufficient state then the dependency is optional; if is it a necessary state the dependency is not optional, but **mandatory.**

It is possible that an action A, say, requires a host of other actions to occur before it can occur itself. If some of these actions have dependencies then some of the dependency to action A may not need to be shown. For example, in the case



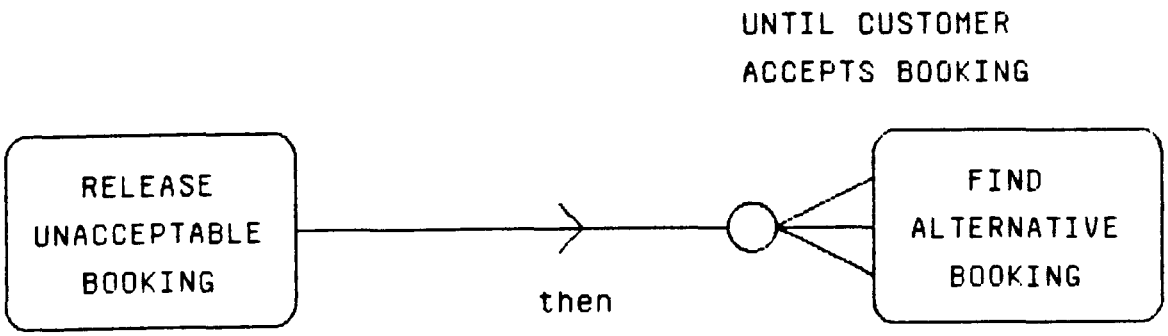
the dependency between B and A is not needed because C needs B to occur before it can occur and A cannot occur before C has occurred. The dependency between B and A is called a **redundant dependency;** A is said to be transitively dependent on B through C. If there are some optionality considerations then the situation is not so simple; the dependency between B and A is only redundant if it gives exactly the same result as the transitive dependency **at all times.**

What we are discussing here is a sequence of actions, B occurs before C before A. However, a principle which must be observed is that actions must be considered to occur in parallel unless some sequence can be shown; i.e. parallelism is a default in action modelling. This is a reversal to most techniques of behavioural modelling.

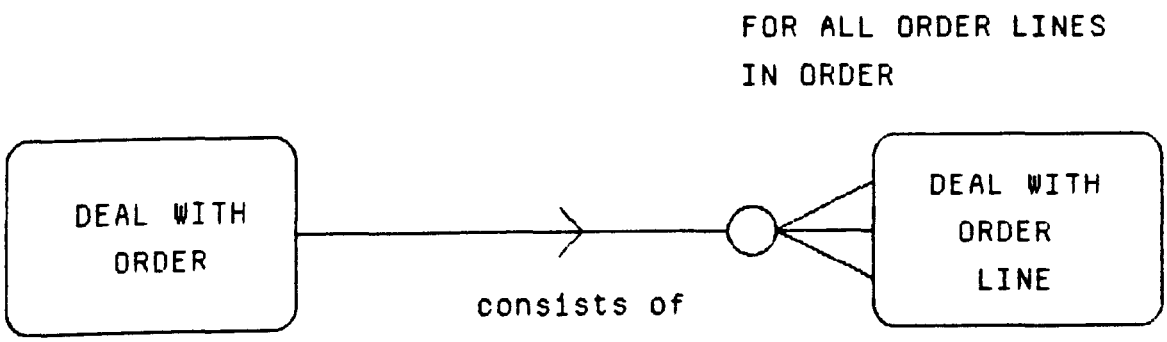
B3.2.2 Cardinality

A further concept concerned with action dependencies is the cardinality of the dependency, ie. whether an action occurs one or many times compared with a related action. As in entity relationship modelling, cardinality is shown by a crow's foot (see figure B3.1). There are two constructs associated with cardinality, 'For All', and 'Until', ie. an action occurs For All members in a set and an action occurs Until 'something happens', as in figure B3.4. The members in a set can be all occurrences of an entity type, all those occurrences of an entity type related to another entity type, or all members of the previous sets which meet a specific condition. For example, all authors, all authors of selected paper X, or all authors of selected paper X who have not yet been invited, respectively. For All should be used wherever possible to avoid any assumption of ordering in a set of occurrences. The only cases where 'Until' should be used are those where an action must be repeated until some condition occurs which is based on a human

or external-body decision. Conditions are considered in greater depth in B3.4. In action modelling the occurrence set and/or condition used is written beside the crow's foot, see figure B3.4.



(Release an unacceptable booking then find an alternative booking until the customer accepts it)



(Dealing with an order consists of dealing with all order lines in that order)

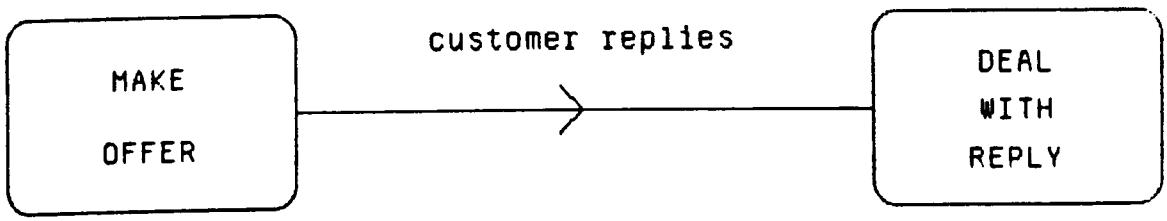
Figure B3.4 Examples of Cardinality

Action dependencies enable the basic design of the structure of procedures. For example, if two actions are associated then the equivalent procedure steps should also be associated. The optionality of a dependency will determine if one step must occur, or may occur, after the other. The cardinality of a dependency will determine if there need to be loops in a procedure and, if so, the form of loop will be known (ie. whether a For or an Until is needed).

B3.3 EVENTS AND TRIGGERING

Events are either the interactions of the area under investigation (the business area) with the external world (which could itself be the subject of an investigation), or they are due to the passage of time, eg. end of month, or they are due to occurrences of actions, eg. Flight Booked [ELLIS, 82]. Nominally an analyst can 'choose' an event from a set of events which all achieve the same ultimate result, eg. if "End of Month" is chosen over "End of Week" then processing is done for all the encompassed end of weeks as a whole with the same ultimate result. So to overcome what is basically a design issue, where it is necessary to have a time event the definition of time should be as loose as possible. For example, "Processing Time Arrived" instead of either "End of Week" or "End of Month". Of course external time events can be specified exactly because the timing is outside the control of the business area, eg. "Tax Year End".

A 'behaviour pattern' is a set of associated action occurrences as determined by the states of an enterprise's entity type occurrences when the pattern is formed. The behaviour pattern is triggered by an event. After triggering, interactions to the external world cause other events to occur. These events are an integral part of the behaviour pattern, for example, requesting a response from a customer. Their existence is wholly dependent on the boundaries of a business area rather than necessity, so they have a different nature to behaviour triggering events. It is important that the nature of an event is distinguished as this has an impact on the design of resulting information systems; for example, a triggering event will have an equivalent if system boundaries change, but a system boundary event may not have. In fact, a system boundary event can be considered to be an action dependency with the interaction as its name, eg. figure B3.5, but triggering events are shown as arrows, see figure B3.1 and figure B5.2.



Make an offer to which the customer replies (goes outside system boundaries implying interfaces in a designed system in a single analysis transaction) then deal with the reply.

Figure B3.5 Example of an Event Dependency

B3.4 CONDITIONS

B3.4.1 Pre and Post Conditions

There are two types of condition which govern the occurrence of a behaviour pattern, **pre-conditions** and **post-conditions** [GOWI,80]. Pre-conditions are enquiries on the state of an organisation to see if an associated action should occur. Post-conditions are enquiries on the state of an enterprise after an action has occurred and enable the determination of which of a number of possible actions happen next. These conditions affect the logic of the action dependencies, ie. they shape a behaviour pattern. The logic of these conditions, which is often complex, can be described in a number of ways. The easiest method is textual, but pseudo-logical methods can be used to avoid ambiguity, eg. predicate calculus [KOWA,79].

Condition logic is associated with a condition, however a given condition can be represented by condition logic in a number of ways; ie. the condition represents the semantics, and the logic the syntax. For example, condition "A & B" is the same as the condition "NOT A OR NOT B", but there are two different condition logics.

Condition logic is placed beside optionality circles on relationships affected by those conditions. Post-conditions are placed at the 'start' of dependencies beside the actions they

are associated with (remember dependencies are uni-directional). Similarly pre-conditions are placed at the 'end' of dependencies beside the actions they are associated with (see figure B3.1). If the condition logic is repeated elseswhere then it can be referenced by a circle containing a number or letter, as in figure B5.2. This highlights duplicated condition logic.

B3.4.2 Exclusivity

There are many cases where only one of a number of dependencies can apply. This is equivalent to 'If-Then-Else' or 'Case' and is called **exclusivity** (see figure B3.1). The conditions for deciding which actual dependency applies to an action occurrence are indicated by circles next to each exclusive dependency, with the condition logic written beside the circles, again see figure B3.1.

B3.4.3 Floating Conditions and Actions

In some cases an action can be triggered by the organisation being in a particular state at any time. This is similar to an event (see B3.3) but has a subtle difference. An event is a happening of a specific kind, a condition is a complex ascertaining of the state of an enterprise which may, or may not, include an event. This type of condition is called a 'floating' condition; one which is not directly associated with an explicit dependency, it 'floats'.

Floating conditions give rise to **floating actions** - actions which have at least one implicit dependency. Floating pre-conditions are represented by a circle on the edge of an action box with no dependency attached to it, eg. Cancel Booking in figure B5.2.

Floating post-conditions are represented by an arrow leading away from the floating action; if it is optional then there will also be a circle at the start of the arrow. It is not possible to apply the concept of optionality to floating pre-conditions because there is no associated activity for it to be mandatorily or optionally dependent on. However with post-conditions it is the action itself which determines whether or not a particular situation must or may apply. Due to this, floating post-conditions often become just a statement of the state of the enterprise at the end of an action instead of a statement about what should happen next.

Floating conditions are equivalent to non-procedural dependencies; ie. there is no explicit procedure, whereas action dependencies have an explicit sequence and hence, are procedural by nature. The instant that a floating pre-condition is fulfilled, the associated action will occur. This is the equivalent of an Artificial Intelligence concept called a 'Demon', a demon being something that is spontaneously invoked when a specific condition is fulfilled (its data state pattern

occurs) - "when its (data state) pattern occurs the demon takes some action ... the result of which may become part of a (data state) pattern that triggers other demons" [SOWA,84].

B3.4.4 Design Justification For Condition Specification

Conditions and condition logic provide the execution conditions which drive a procedure. Every branch and loop in a procedure should have an execution condition to determine what to do; if there is no condition then the default action would be taken (whatever that was). Floating actions enable non-procedural, implicit dependencies to be dealt with. In effect floating actions can accomplish the complete modularisation of every action. If this is designed in a system that can cope with non-procedularity then every procedure or procedure step would be completely modular, yet implicitly linked where necessary. As described in part B7.2, the communication of a specification diagram can also be improved by such concepts as floating actions.

B3.4.5 Premises, Conclusions and Probabilities

Behaviour pattern types (generalisations of behaviour patterns) establish **premises** (a premise is a statement about the state of an enterprise). The premises established by a behaviour pattern type are shown as post-conditions from the 'final' actions in the action model representing a behaviour pattern

type. A premise may have more than one condition associated with it, eg. 'If-Then-Else' cases and where a premise has more than one part. Premises are the basis for reaching conclusions about the state of an enterprise, and enable the confirmation of other premises. If a conclusion can be drawn from a particular premise, the conclusion is written alongside a dependency beside the post-condition which shows the establishment of the premise on which the conclusion is based.

Traditionally in information systems conditions are either fulfilled or not. However we know from knowledge-based systems that conditions may be fulfilled according to a previously calculated probability (eg. fuzzy logic [ZADEH, 78]). If a conclusion can only be reached with a given probability then this is associated with the conclusion, again see figure B3.1. In the case of floating post-conditions any premises, conclusions and probabilities are written along the arrow emanating from the action which caused them to be drawn. If a probability is required in the establishment of a premise then it will be incorporated into any pre-condition logic that refers to it.

Probability and conclusions were introduced to cope with the differing requirements of knowledge representation over information systems representation; they are the only new conventions which had to be introduced. There could be a good case for applying these concepts to information systems both in

entity relationship modelling and in action modelling.

B3.5 BEHAVIOURAL HIERARCHIES

Behaviour can be depicted with a hierarchy formed either by abstraction or by decomposition. It is almost essential to make use of hierarchy for management and comprehension purposes in behaviour modelling due to the large quantities of information gathered.

In action modelling there are strict rules concerning hierarchy formation which tie the hierarchy closely to the entity types whose behaviour is modelled.

We are interested in decomposing activity into lower-level activity. We are also interested in decomposing data into smaller components - subject areas into logical horizons into entity types into attributes. The rule for decomposing actions is that it must follow the decomposition of data: this is really a stringent case of the notion put forward by Jackson for structuring programs [JACK, 75].

So an action must be on a logical horizon, on an entity type, or on an attribute; actions on logical horizons decompose into actions on entity types, which in turn decompose into actions on attributes. For example, see figure B3.6.

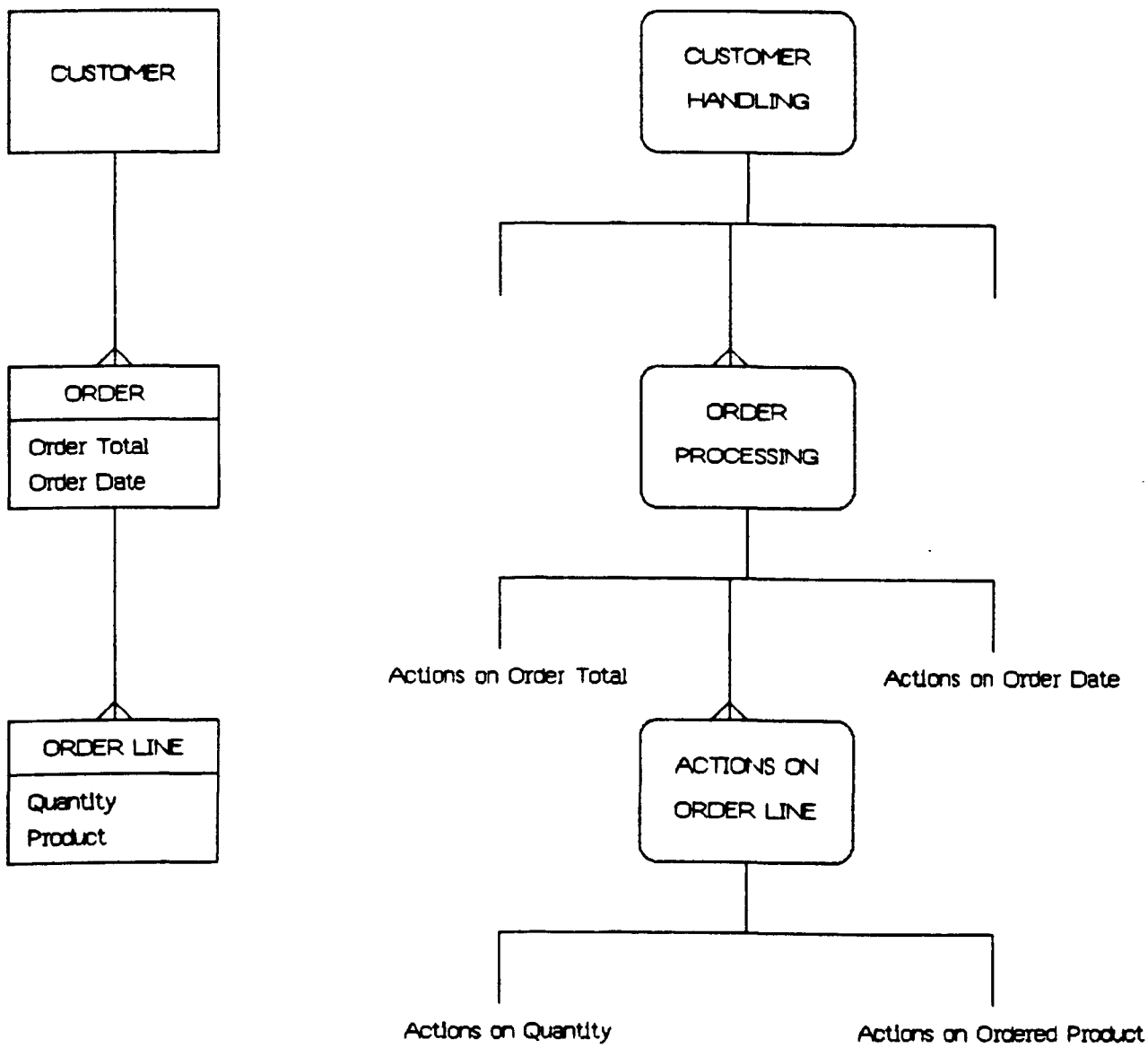


Figure B3.6 Example of Processing Decomposition Based on Data Structure

This is considered further in section B4 and chapter D.

The common method of coping with hierarchy of behaviour is to modularise and form sub-models, the sub-models being aggregated into the decomposed object.

In action modelling there are two other conventions for depicting hierarchy; these cater for different situations and for the restrictions of a two-dimensional media. One convention shows the hierarchy by dependency (see figure B3.1). However complex hierarchies can be difficult to draw and comprehend in this manner, especially when there are more than two actions in a hierarchy. The other convention deals with this and is the equivalent of the sub-entity type convention; it effectively results in a sub-model, again see figure B3.1.

B3.6 DUPLICATION OF USE

As in entity relationship modelling, actions should only appear once in a model but be referenced whenever necessary. This is in accord with the basic principles of the modular use of software, but is rarely applied to behaviour specification. The depiction of shared usage is very difficult and can result in some of the communicability or the purity of any diagrams being undermined. The representation of duplicated usage by dependency may result in large numbers of dependencies, causing problems for the

production and maintenance of diagrams. This is overcome by the controlled duplication of the action (eg. by the use of // to indicate duplication), with a consequential loss of purity of the diagrams.

B3.7 INDIVISIBLE ACTIONS AND SELECTION CRITERIA

The majority of the concepts discussed in this section are pure BAA concepts, ie. are only concerned with specifing requirements. However there are 'indivisible actions' (as opposed to elementary actions) which are much more design-oriented. An indivisible action is an action which cannot be broken down any further. There are a number of different types of indivisible action, the allowable types varying with the type of data model being manipulated. Every data type must have a create and delete manipulation (in some models, eg. semantic hierarchies, the actual creation of some data occurrences can result from the requirements gathering process). There should also be a wide range of modifications, retrievals, and indivisible I/O actions. Relationship manipulations are needed for entity relationship modelling; other data models have different specialised constructs (eg. relational algebra has join & projection [CODD, 70]). Indivisible actions are derived from a data model's data manipulation language, if it has one, and are mainly elicited for design purposes. Entity relationship models have ESTABLISH, UPDATE and SELECT entity type, REFERENCE and MODIFY attribute,

and ASSOCIATE, TRANSFER and DISSASSOCIATE relationship.

The manipulation of data requires the identification of the occurrences to be manipulated. Thus some form of selection criteria is essential. Selection criteria isolate the occurrence, or set of occurrences to be manipulated; for example 'manipulate each thing where thing is a male person' has the effect of manipulating each male in the area under manipulation. A retrieval can be made before every manipulation and the latest retrieved occurrence manipulated (eg. the notion of currency in Codasyl Databases [CODA, 71]). Alternatively, every manipulation can have selection criteria to isolate the occurrence(s) to be manipulated (as in query languages such as SEQUEL [CHBO,74]. These should be able to be combined to achieve the desired effect depending on the data model rules. Exact selection criteria are needed for every level of action. However indivisible actions require more specific selection detail than higher level actions, which can have broader criteria specified. The detail is necessary for design, but as it tends to be confusing to users, it is not suggested for user requirements specification.

B3.8 INFORMATION FLOW

It is possible to define a flow of information on an action dependency. Where an action produces information which is required by another, this will show up as a dependency. In this

case the information should be written above the dependency line. An information flow in this fashion is identical to a conclusion as described in B3.4.5. In fact conclusions are one form of information flow; note that the representation is the same.

B4 ANALYSIS OF ACTIONS

B4.1 INPUTS TO ACTION ANALYSIS

Action Analysis requires that a decomposition diagram and a dependency diagram are available which include the process under investigation, which is usually an elementary process, but not necessarily. A definition of the process and its expected inputs and outputs is also necessary (the inputs and outputs can be found from the dependency diagram).

An entity relationship model, or appropriate section of it is required so that the logic of the process can be matched with the entity types and their predicates.

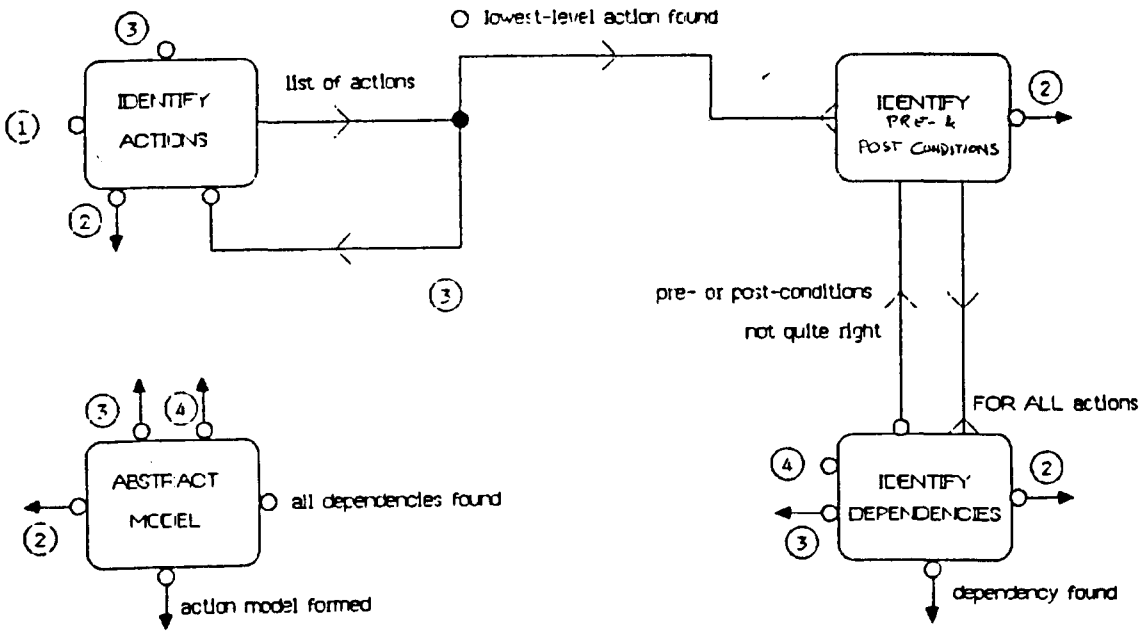
Access to a user would be beneficial to enable the required inherent logic to be elicited "from the horse's mouth".

B4.2 OUTPUTS FROM ACTION ANALYSIS

Action Analysis produces an action model consisting of an action dependency diagram and action descriptions.

Action Analysis may also produce changes to the inputs in the form of improvements due to the much better understanding of the process under analysis.

B4.3 ACTION MODEL OF ACTION ANALYSIS TASKS



(1) decomposition diagram & dependency diagram & process decompositions & entity relationship model & access to user available

(2) change to input found

(3) need to identify more actions

(4) more work needed on finding dependencies

Figure B4.1 Action Model of Action Analysis Tasks

B4.4 ACTION ANALYSIS TASKS

B4.4.1 Identify Actions

First the actions of the process need to be identified. This is best done by considering each output in turn and figuring out how it can be produced from the input, given the constraints of the entity relationship model.

For example, if one output from a Produce Order process is an order then the required actions may be:

FIND ORDER REQUIREMENT FOR A MATERIAL

FIND BEST SUPPLIER FOR THAT MATERIAL AND QUANTITY

PREPARE ORDER

SEND ORDER

Note that this assumes single material orders. If Purchase Orders were required to be batched up by some means then this list would be more complex.

B4.4.2 Decompose Actions Where Necessary

If some of the actions identified in B4.4.1 are not elementary,

ie. involve more than one entity or entity type, or are thought to be fairly complex, eg. involving a number of decisions, then decompose them following the rules in B3. This is really a repetition of task B4.4.1 using the actions that result from it.

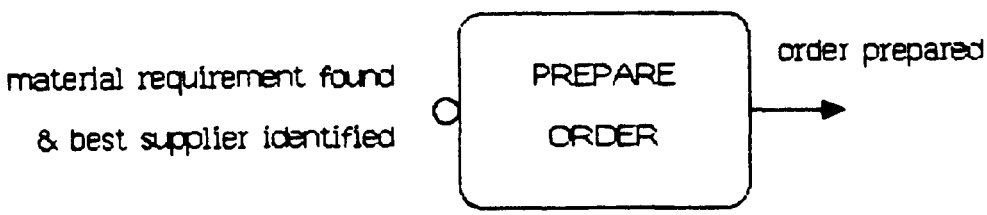
For example, Find Order Requirement For A Material may involve examining current stocks of a product and current requirements for it and deciding how much to reorder for it based on standard reorder levels, etc. This action can be seen to involve a number of entity types and decisions, so is not elementary.

Definitions of elementary actions must be produced at this time.

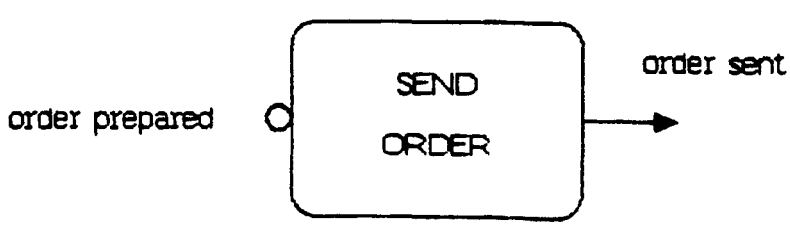
The question arises of what to do with the elementary actions on the lowest level of decomposition. We can decompose further to indivisible actions; if so these should be described in an action diagram [MART, 85a] or other form of structured language, because we are now designing the detail of the action not analysing requirement (see appendix X1). An alternative would be to use something like a decision table to describe the required logic of the action. It is just not sensible to draw action dependency diagrams for this level due to the vast volume of detail. It also is not necessary as below this level we are more or less designing, not analysing.

B4.4.3 Identify Pre- and Post-Conditions

Eventually a point will be reached where sufficient decomposition has been done. An action model is now constructed for the actions identified. These actions are at a number of levels; construct a decomposition diagram to show the levelling. Next take the lowest-level actions in turn and analyse them to identify their pre-conditions and post-conditions in isolation, eg.



Prepare Order has pre-conditions of there being something to order and someone to order from and a single outcome of a prepared order.



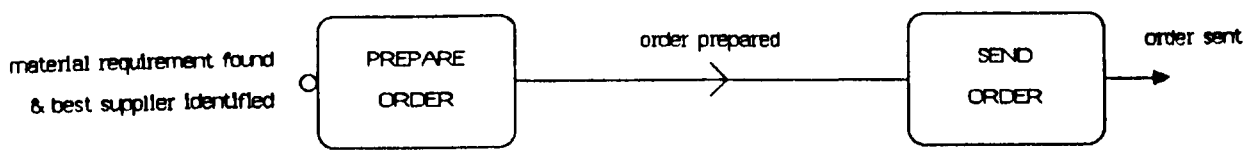
Send Order just takes a prepared order and sends it to the chosen supplier by some means (the means is a design issue, eg. electronic mail, or hand-post, or GPO post).

The optionality of the post-conditions must be considered at this time.

B4.4.4 Identify Dependencies

The result of B4.4.3 is a set of floating actions. Now we need to match up the pre- & post-conditions to identify dependencies.

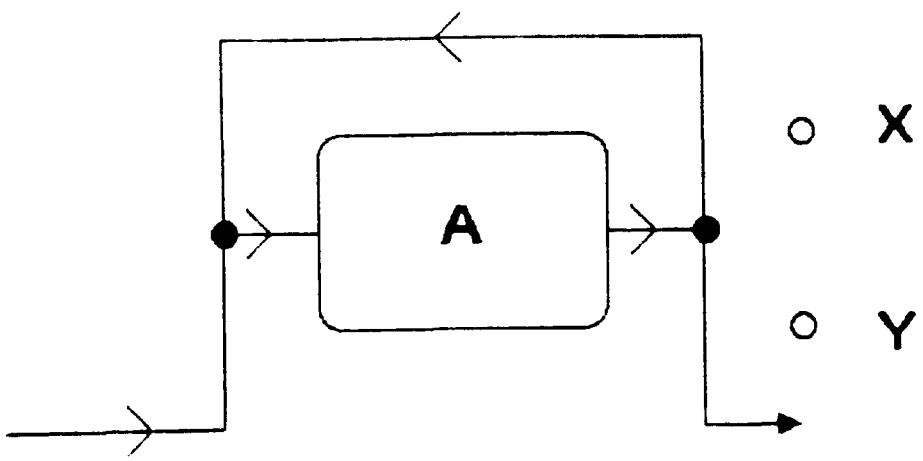
For example, in B4.4.3 Prepare Order had a post-condition of prepared order, which is the pre-condition of Send Order - so there must be a dependency between them, an order cannot be sent until its prepared:



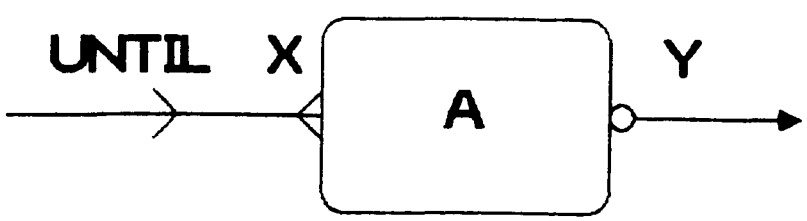
This should be done for all matching conditions; note that care needs to be taken where there is not a simple match, eg. in the case of complex condition logic.

Where a particular condition matches a number of others, the action is best represented as a floating action.

Where a recursive action is found, ie. one with an involuted dependency, this can often be replaced by a crow's foot as it usually indicates repetition, eg.



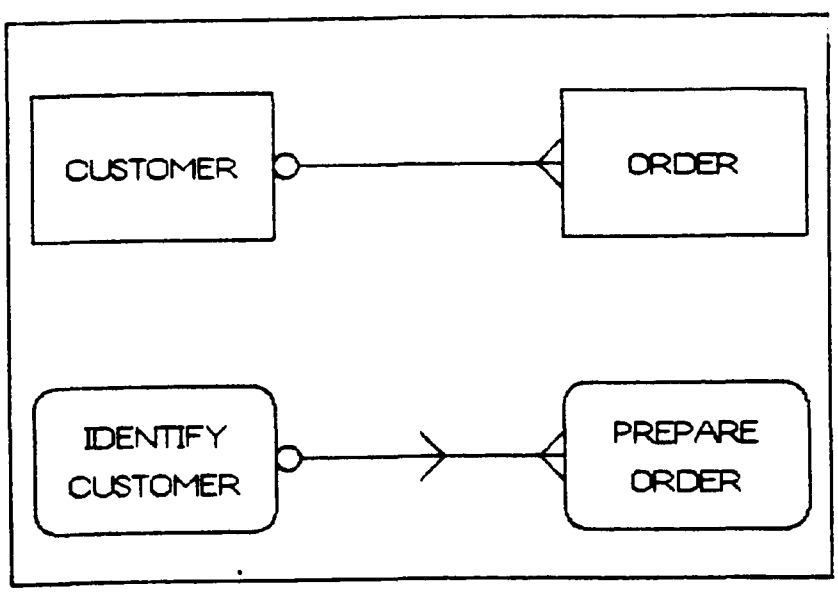
is often the same as:



Where the action is acting on an entity type with an involuted relationship, eg. a bill-of materials, then this must stay as a

recursive action to enable the correct following of the involuted relationship. The basic difference between a recursive and a repeated action is that each occurrence of a recursive action assumes it is for the 'first' time, whereas a repeated action may build on previous occurrences, for instance, by constructing a total. Where this consideration is important, such as following an involuted entity type relationship, careful thought must be given to find the best representation.

Other cases of optional and repeated action can be elicited by inspecting the entity relationship diagram. Where two dependent actions act on two related entity types then the optionality and cardinality of the entity type relationship will apply to the action dependency, eg.



Optional dependencies can be found by considering the condition matching. Optional post-conditions are found during B4.4.3.

They may be changed by the matching process if necessary, though this does not happen often. Where a single pre-condition matches a single post-condition, the dependency must be mandatory as there are no other uses for that pre-condition. Similar considerations can be made for post-conditions, but it is not as simple in this case because the dependent action may not be needed at all. Where a condition matches more than one other exclusively, the same arguments apply because only one condition can be matched at any one time. If there is no exclusivity then the dependency is optional for that condition because there will be times when it is not necessary.

Common sense and user input will also give some optionality and cardinality.

B4.4.5 Abstract Model

Now there is an action model for the lowest-level actions. It is often useful to show higher-lever actions on the same diagram (see B3.5 Behavioural Hierarchies). This can be done for reasons of communication.

Occasionally a dependency applies to a group of actions rather than just one in isolation - if this group forms a higher-level action then the higher level action must be shown with an appropriate dependency.

If a diagram becomes very large it can be split along the lines of the action decomposition, with one diagram per 'leg' of the decomposition.

B4.5 OTHER POINTS

Each of the tasks may cause an iteration of the analysis which will continue until a satisfactory model is achieved.

Changes may be found to the input along the way due to the depth of consideration of the models. These should be dealt with appropriately (eg. by informing the 'guardian' of the model concerned). In practice it is not always necessary to totally follow all the rules and guidelines of action modelling; there just may not be enough time to do so. Useful results can be gained without having a perfect model, after all the technique has to be pragmatic otherwise it will not be used.

Events, premises, conclusions and probabilities should be identified when the pre- and post-conditions are defined, or during the matching process. Events tend to be found where there is a pre-condition with no matching post-condition.

This discussion has not been able to consider every case due to the vast number of possibilities.

B5 EXAMPLES OF ACTION MODELS

It is useful to be able to see examples of action models to help bring home the concepts discussed in earlier sections. This part contains three examples, a Book Holiday process, an Organise Conference process and two rules from MYCIN - an expert system. All of the examples have been simplified to enable discussion of the principles. In general only the minimum of requirement has been catered for and little exception processing has been considered. That said, it is likely that this would apply in practice as well due to the lack of time which often applies at this stage of an analysis process.

B5.1 BOOK HOLIDAY

Figure B5.1 shows the entity relationship diagram for Book Holiday.

Figure B5.2 shows an action dependency diagram of this process.

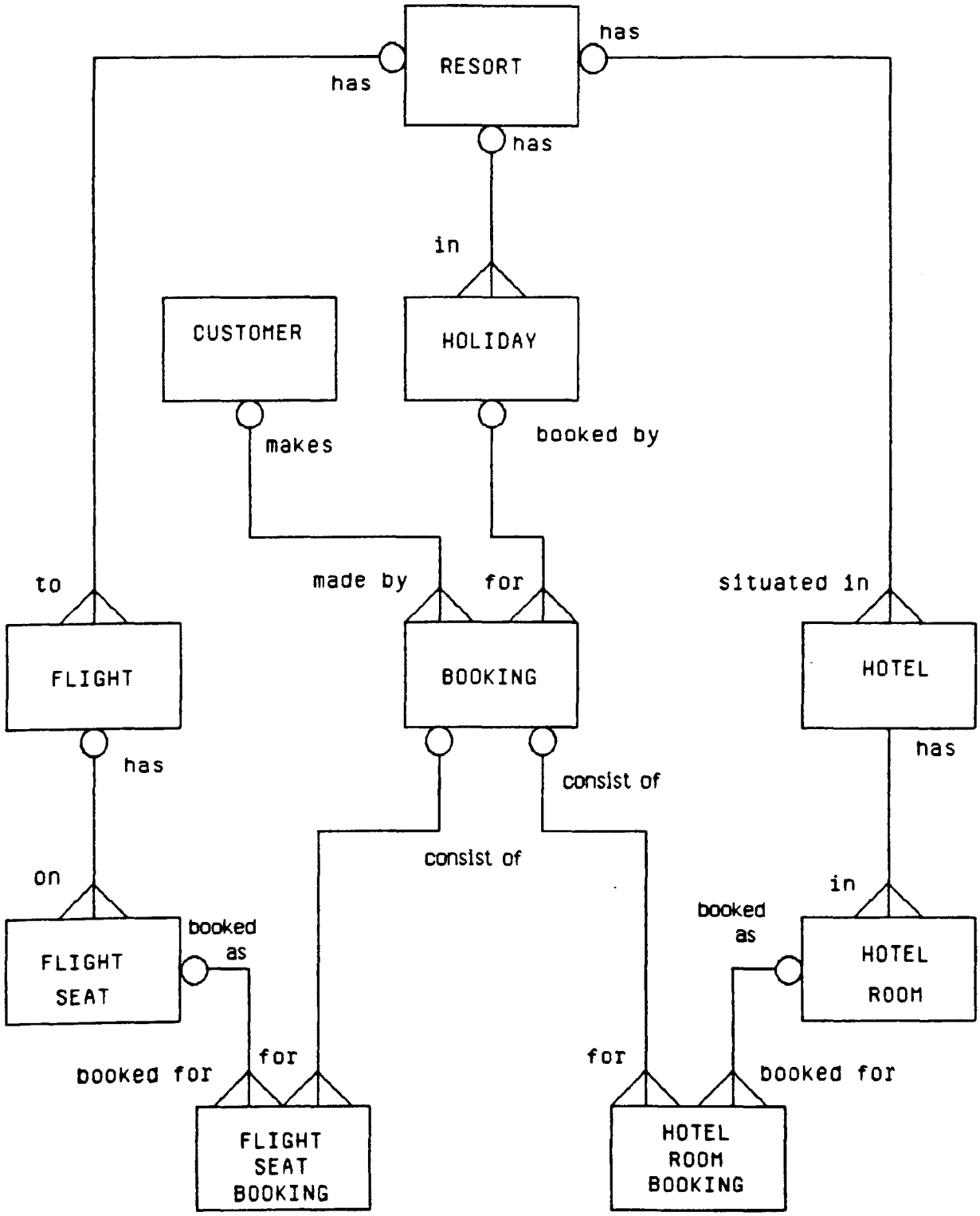


Figure B5.1 Entity relationship Diagram for Book Holiday

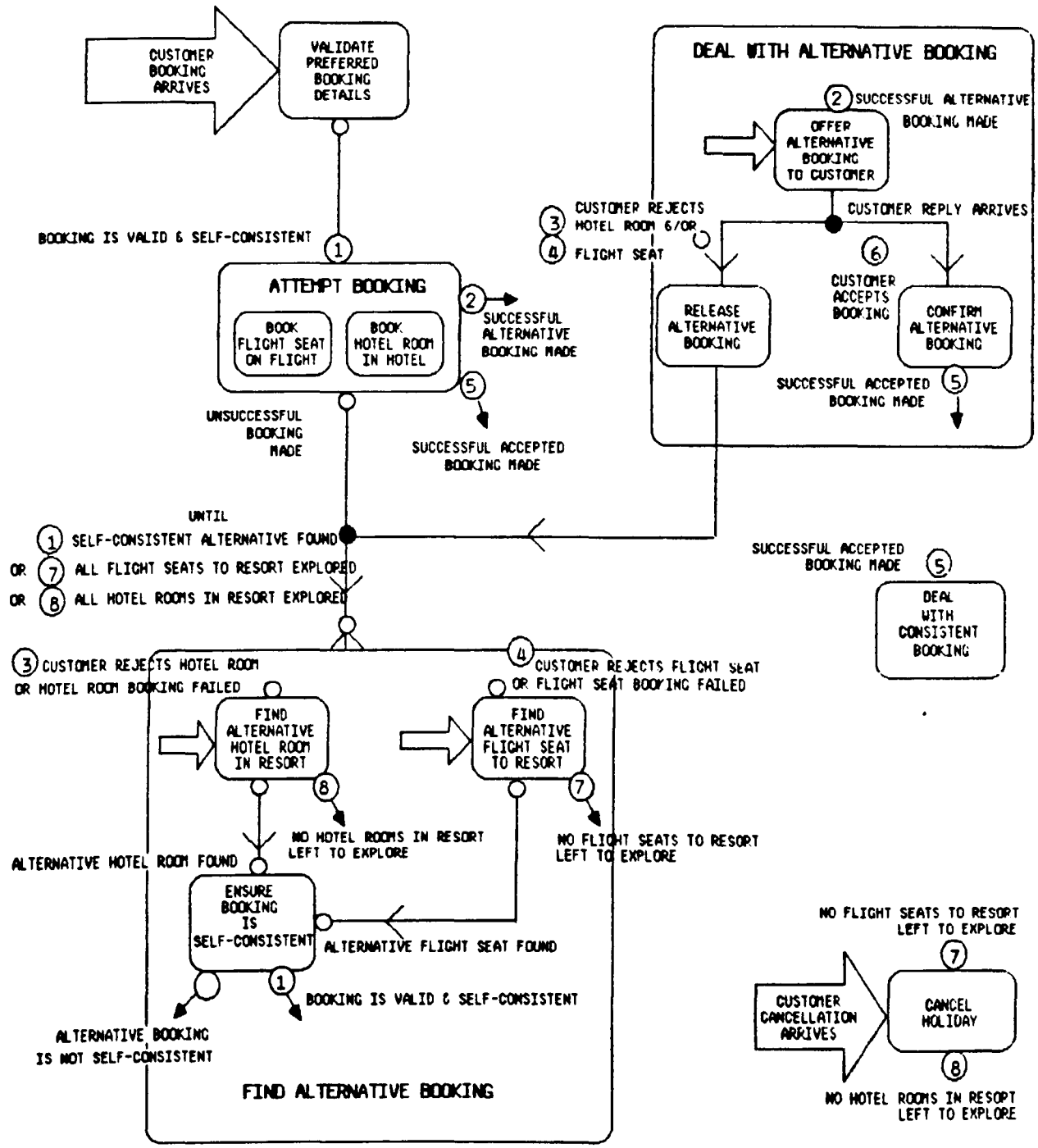
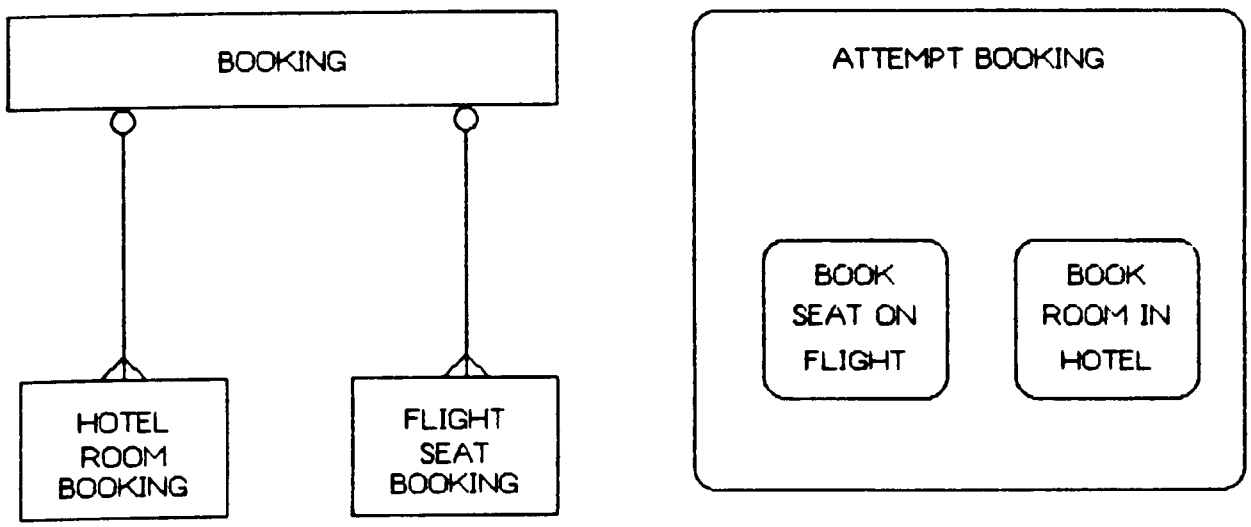


Figure B5.2 Action Dependency Diagram for Book Holiday

If the action model dealing with the booking transaction is examined (figure B5.2), a good place to start is in the top left hand corner, at the large open arrow. The arrow represents a triggering event and in this case that event is an external interaction with a customer. The customer makes a holiday booking which triggers the Validate Preferred Booking Details action represented by the box with rounded edges (soft box). A preferred booking is one that is initially chosen by the customer.

Once a valid booking is available, it is used in the Attempt Booking action. The action dependency between the two actions has a circle at each end indicating optionality. In this case it is fully optional, ie. action Validate Preferred Booking Details need not be followed by action Attempt Booking and action Attempt Booking can occur without Validate Preferred Booking Details occurring. Once the preferred booking is valid and self-consistent, which means that the flight specified is consistent with the hotel and resort etc., then the Attempt Booking action can occur. This condition is labelled ① on the diagram and is a pre-condition for the action to occur. If the condition is not fulfilled and the booking is invalid in some way, a series of interactions with the customer would occur to sort the problem out. These interactions are not shown. The Attempt Booking action consists of two sub-actions, Book Flight Seat ... and Book Hotel Room ... , ie. a hierarchy of actions exists; furthermore these actions can occur in parallel because

no dependency exists between them. Note that the hierarchy matches the logical horizons of the affected entity types in the entity relationship diagram:



If both sub-actions are successful then post-condition 5 is fulfilled. This will trigger action Deal With Consistent Booking on the right hand side of the diagram. This action has condition 5 as a pre-condition, indicated by the circle above the action, and this means that the action can occur at any time that this pre-condition is fulfilled, which is why it appears as a floating action. The fact that it is floating also indicates that it could occur in parallel with other actions. If the attempted booking were unsuccessful then the action Find Alternative Booking would occur, this is the large box in the

bottom half of the diagram and contains a set of interacting sub-actions. The dependency is followed in the direction of the arrow and the crow's foot indicates a one to many relationship to the Find Alternative Booking action. This means that this action can occur many times, the actual number of occurrences is determined by the UNTIL conditions, in this case there are three conditions numbered ①, ⑦, ⑧ and the action Find Alternative Booking is repeated until a consistent alternative booking is found, or until all seats and rooms have been tried. The sub-actions within the action Find Alternative Booking are started at either of the open arrows depending on whether it was the hotel or the flight booking which was unsuccessful in the previous action. The booking is then checked to be consistent and a variety of post-conditions exist which may or may not fulfill the UNTIL conditions. If a booking is valid and self-consistent this terminates the Find Alternative Booking action and triggers the Attempt Booking action again. This may be unsuccessful in which case the Find Alternative Booking action is initiated again, or post-condition ② is fulfilled (as here we are dealing with an alternative booking rather than a preferred one) in which case the Deal With Alternative Booking action situated in the top right hand side of the diagram is triggered as the pre-condition or a successful alternative booking has been fulfilled. The customer is then offered the alternative booking. Assuming the customer replies at some time (interaction event as dependency), one or other of

the dependences depicted with the mutual exclusion symbol (filled in circle) is followed. The customer either accepts it and the booking is confirmed resulting in the triggering of the floating action Deal With Consistent Booking, or he rejects it, the booking is released and the dependency indicates that Find Alternative Booking is initiated again. This continues until an acceptable booking is found or there are no alternative bookings left to explore (see floating action at bottom right of the diagram).

As will be discussed in section B7.2, it is possible to describe action models in totally procedural terms, totally non-procedural terms and, the best, a compromise - procedural where necessary and non-procedural where necessary. Figure B5.2 shows the compromise case for Book Holiday. The next two figures cover the totally procedural and totally non-procedural cases for this process.

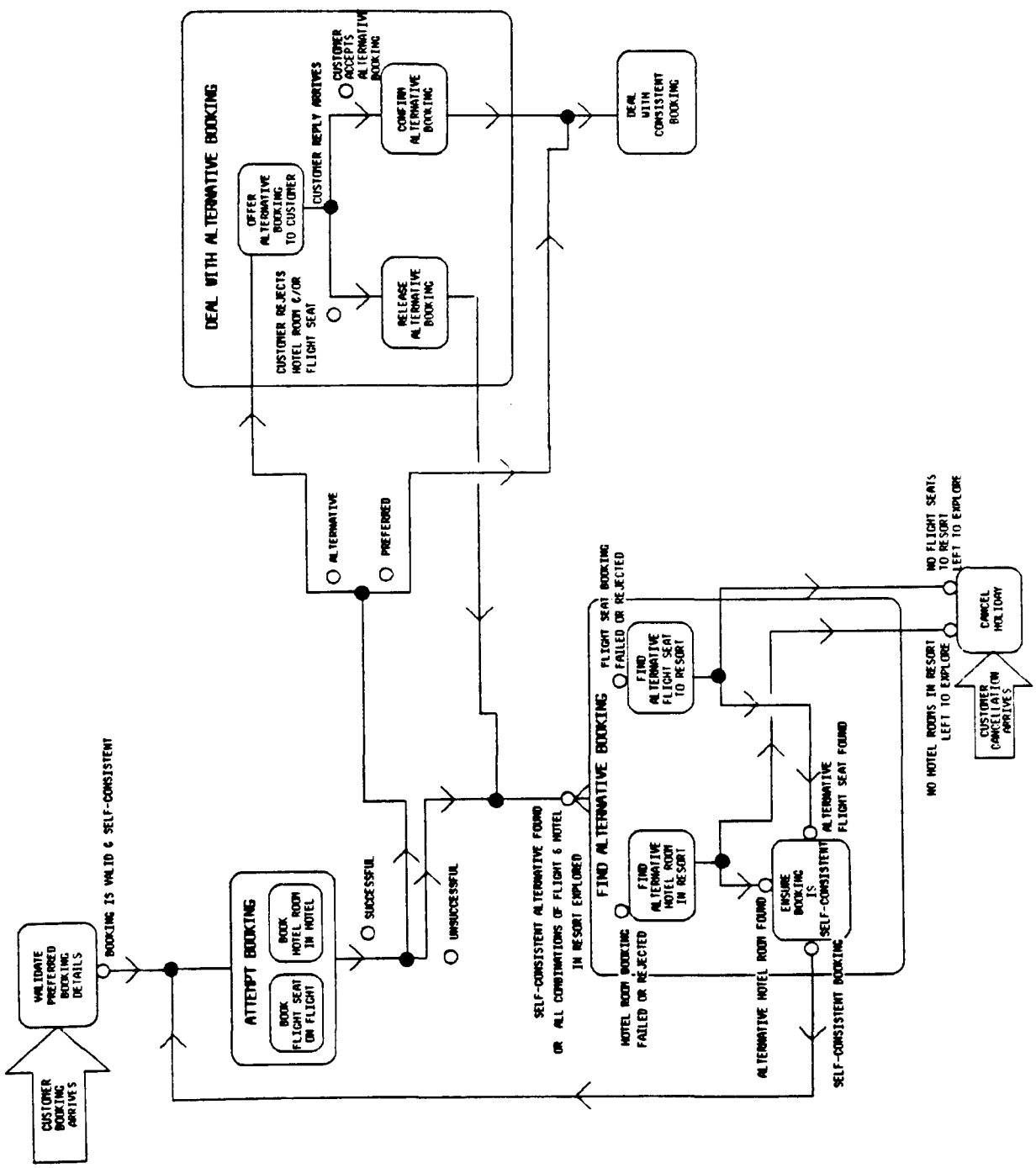


Figure B5.3 Procedural Diagram of Book Holiday

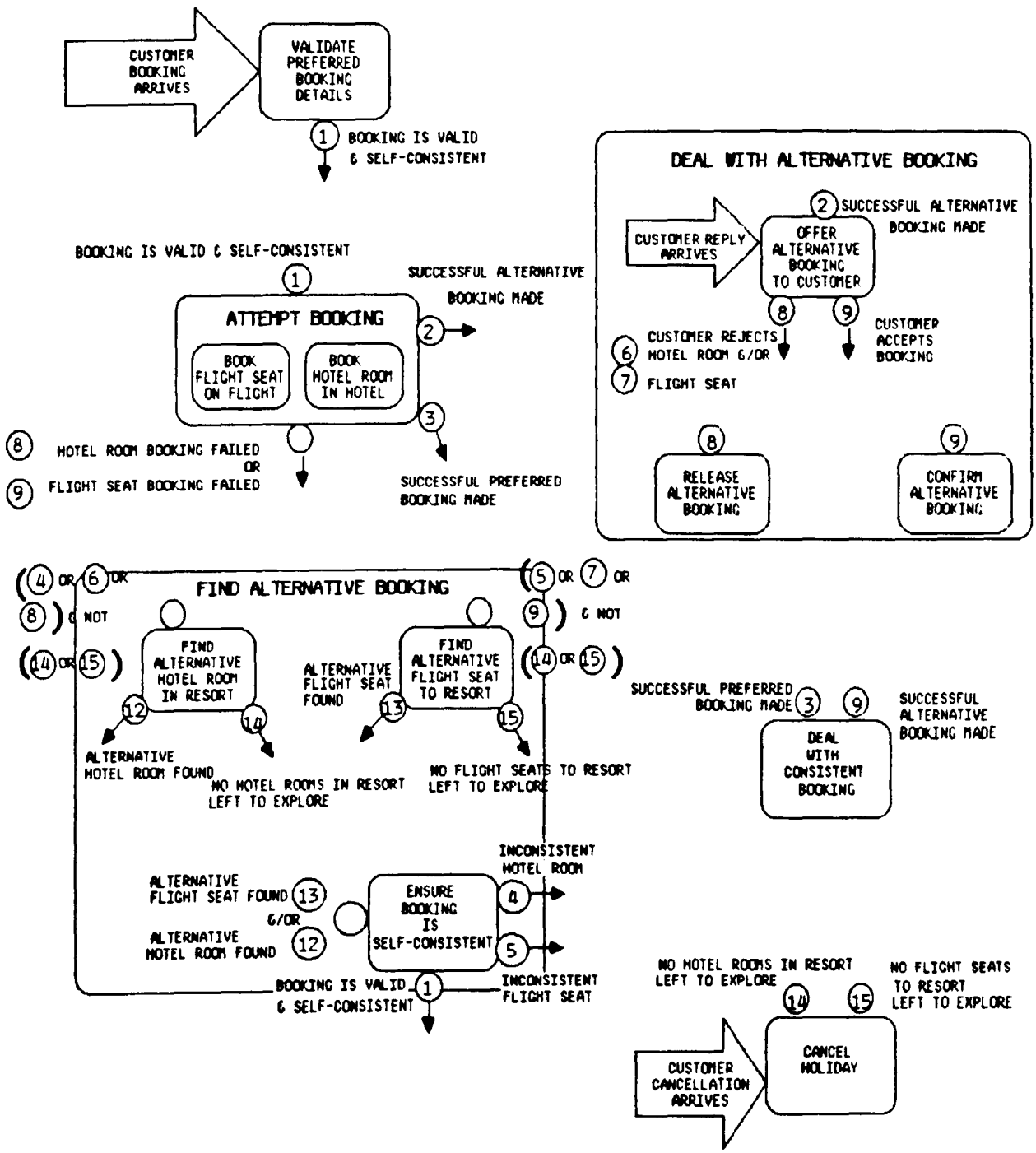


Figure B5.4 Non-Procedural Diagram of Book Holiday

B5.2 ORGANISE CONFERENCE

Figure B5.5 shows the entity relationship diagram for this example.

Figure B5.6 shows the process decomposition of the area.

Figure B5.7 shows an action dependency diagram of this process.

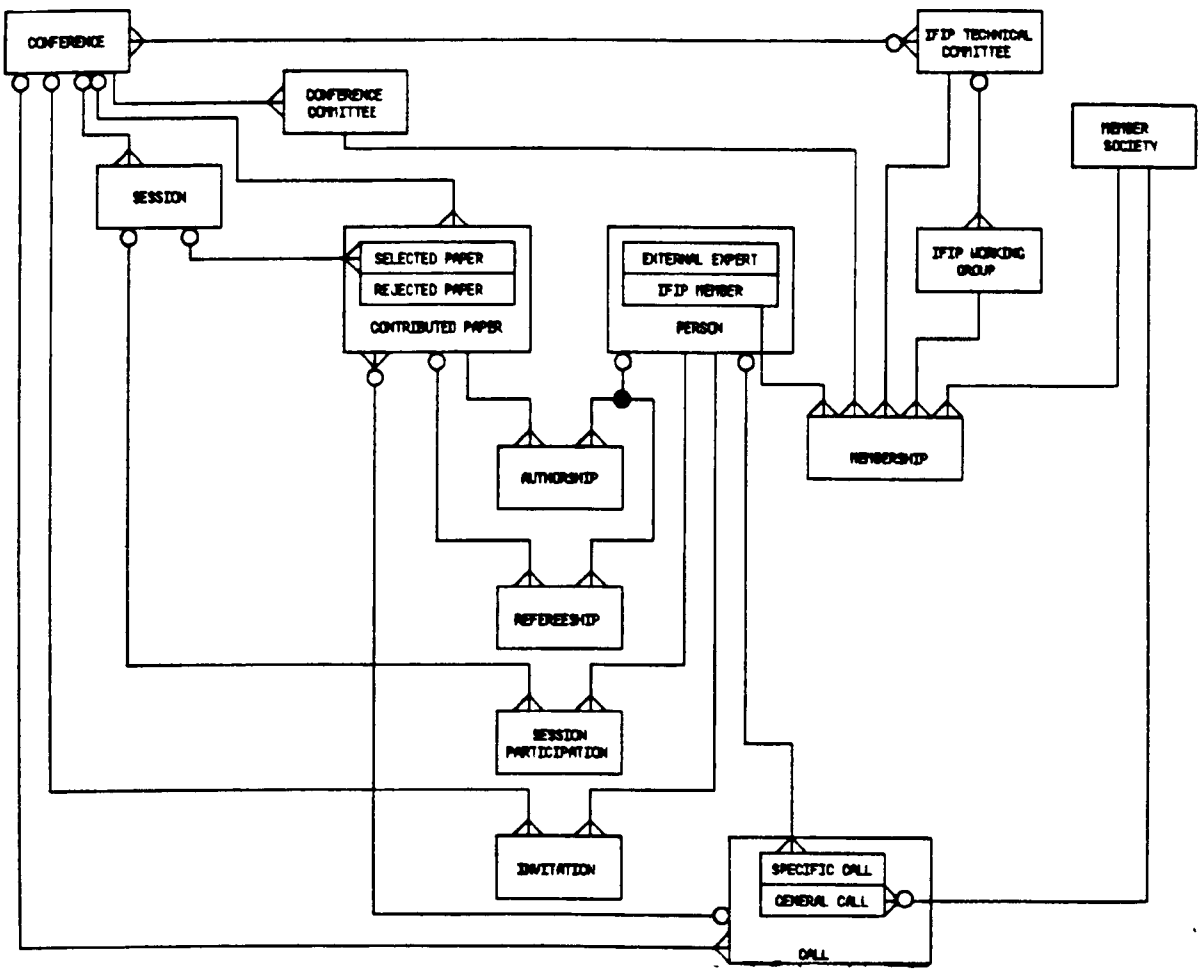


Figure B5.5 Entity Relationship Diagram for Organise Conference

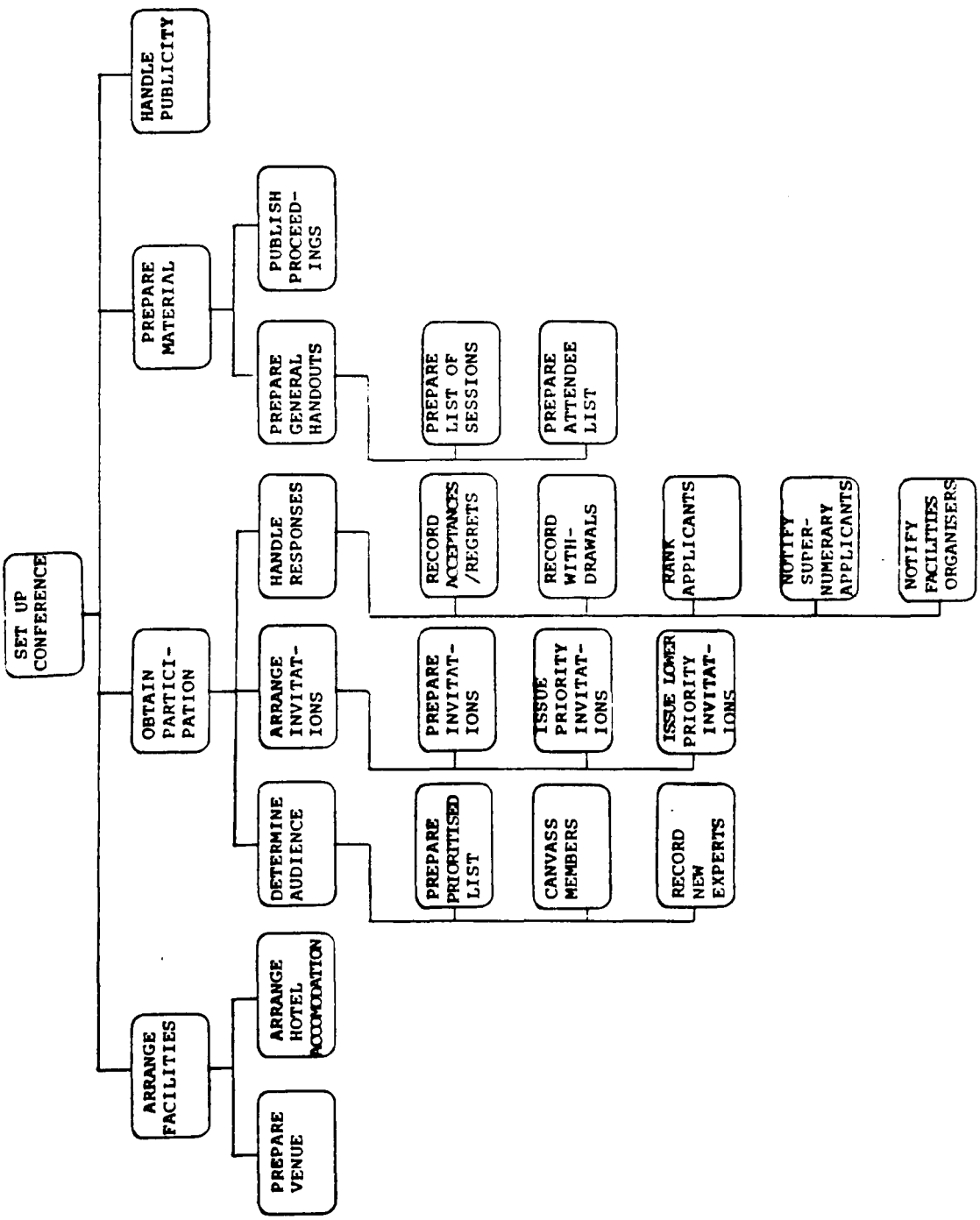


Figure B5.6 Process Decomposition of Conference Organisation

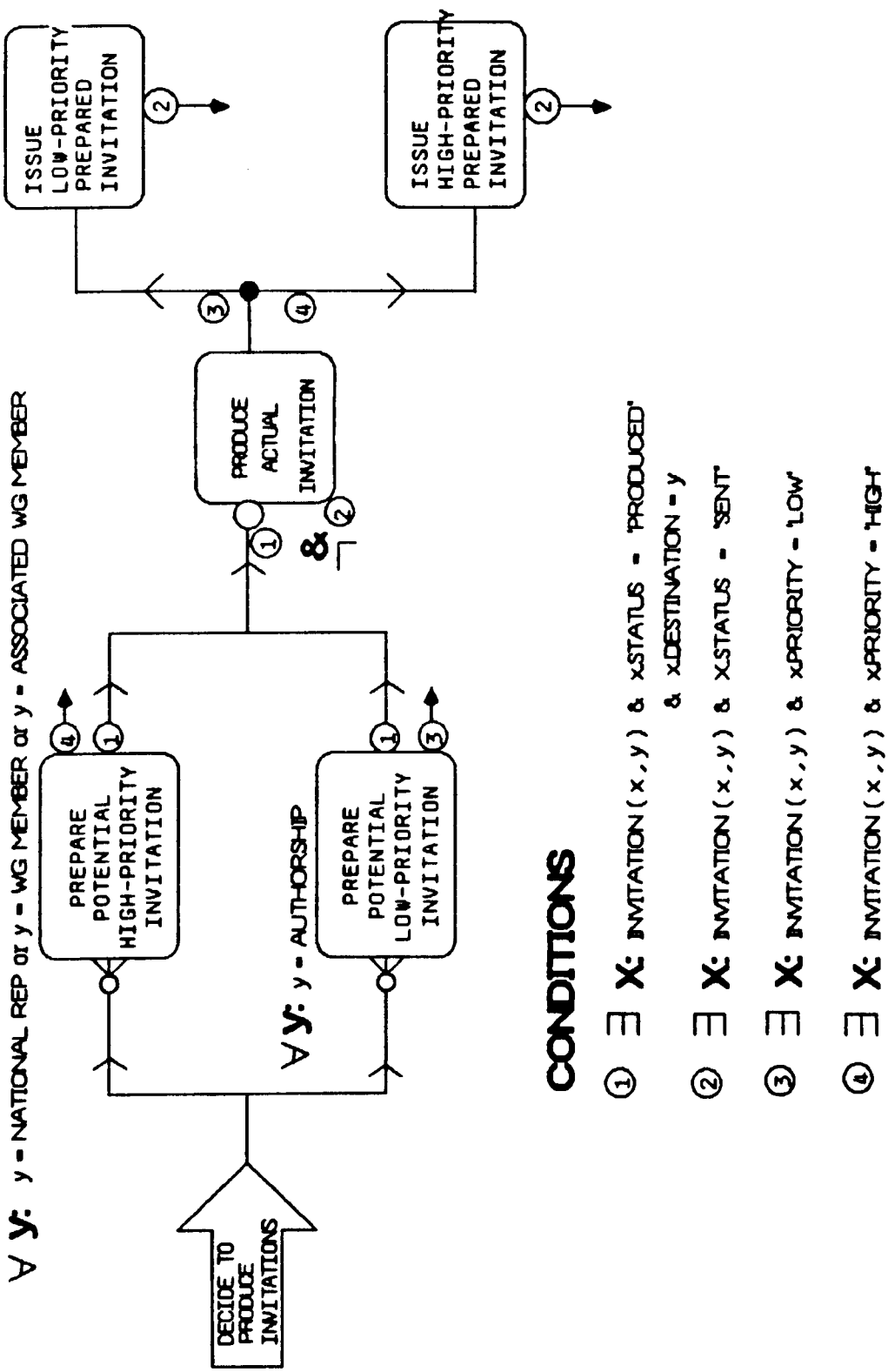


Figure B5.7 Action Dependency Diagram of Organise Conference

This example is based on the IFIP case study used in the CRIS conferences. A description can be found in [OLLE, 82]. The example is concerned with the activities of the organising conference. These were described as:

1. Preparing a list of people to invite to the conference

2. Issuing priority invitations to National Representatives, Working Group members and members of associated Working Groups

3. Ensuring all authors of each selected paper receive an invitation

4. Ensuring authors of rejected papers receive an invitation

5. Avoiding sending duplicate invitations to any individual

6. Registering acceptance of invitations

7. Generating final list of attendees

Here we will only be concerned with activities 1 through 5.

As this example is similar in aspect to that described in part B5.1, a detailed explanation will not be given.

The functional specification for this example is taken from [MACP, 82], see figure B5.6. The particular processes we are concerned with have been highlighted, these being: Prepare Prioritized List, Prepare Invitation, Issue Priority Invitations and Issue Lower Priority Invitations. Figure B5.5 was also taken from [MACP, 82], but the conventions have been updated to those of Information Engineering. Again the entity types we are concerned with have been highlighted, these being: Invitation, Person, Contributed Paper and Authorship. Further detail is required concerning the entity types, for example, their attributes and definitions. However these are not very important to behavioural specification, so have not been shown.

Figure B5.7 links the identified processes with the identified data. If figure B5.7 were executable, its results would be an optimal list of invitations. Figure B5.7 represents my interpretation of the business rules behind preparing such a list. It assumes that the organisation is 'inconsistent' between deciding to invite someone and actually issuing the invitation. The form of the diagram is such as to minimise the time of inconsistency. Different business rules would produce a different diagram. The diagram covers the full spectrum of activities 1 to 5, and is a complete specification of the behaviour at an elementary action level. It is possible to

continue to a lower level of detail, but it is considered inappropriate both for this example and for user requirements specification in general.

The concepts covered by the diagram are elementary process (the complete diagram), the event which triggers it, actions, relationships, cardinality (the initial 'For Alls'), optionality, conditions (four of which are numbered), condition logic (written in predicate calculus), and floating post-conditions. There is little about inviting people which is not covered by this diagram. As already mentioned, depth can be added (eg. by specifying the detail of the action by predicate calculus), but it is felt inappropriate to do so for a user requirements specification.

B5.3 MYCIN RULES

Action modelling can be used in knowledge aquisition to describe rules for a knowledge base [FEFI, 85b]. The entity relationship diagram describes facts. This is discussed further in section B7.4. However an example has been included here. This example concentrates on two rules from the MYCIN expert system [SHOR, 76].

The MYCIN rules which are depicted are:

```
IF  THERE IS AN ORGANISM REQUIRING THERAPY OR
    THERE IS EVIDENCE OF THE EXISTENCE OF ADDITIONAL
    ORGANISMS
THEN
    CONSTRUCT LIST OF SUITABLE THERAPIES FOR AN ORGANISM
    AND DETERMINE THE BEST THERAPY FOR A PATIENT
OTHERWISE
    INDICATE THAT THE PATIENT DOES NOT REQUIRE THERAPY
```

and:

```
IF  THE IDENTITY OF AN ORGANISM IS KNOWN AND
    THERE IS AN IDENTIFIABLE DISEASE FOR THE ORGANISM
THEN
    THERE IS DEFINITE EVIDENCE (PROBABILITY OF 1.0) THAT
    THERE IS AN ORGANISM REQUIRING THERAPY
```

Figure B5.8 shows the entity relationship diagram for the base facts knowledge domain.

Figure B5.9 shows the action dependency diagram for the two rules.

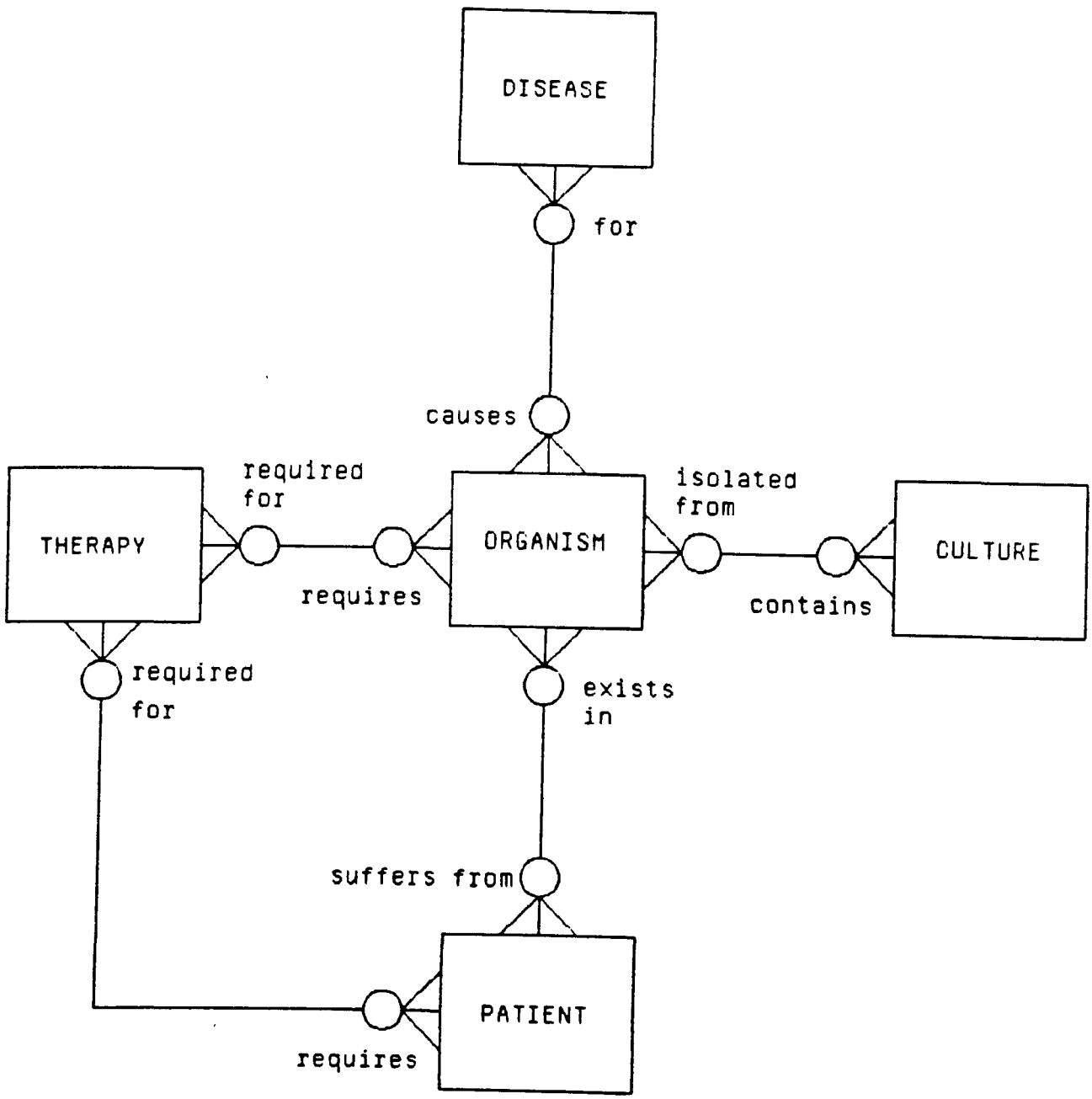
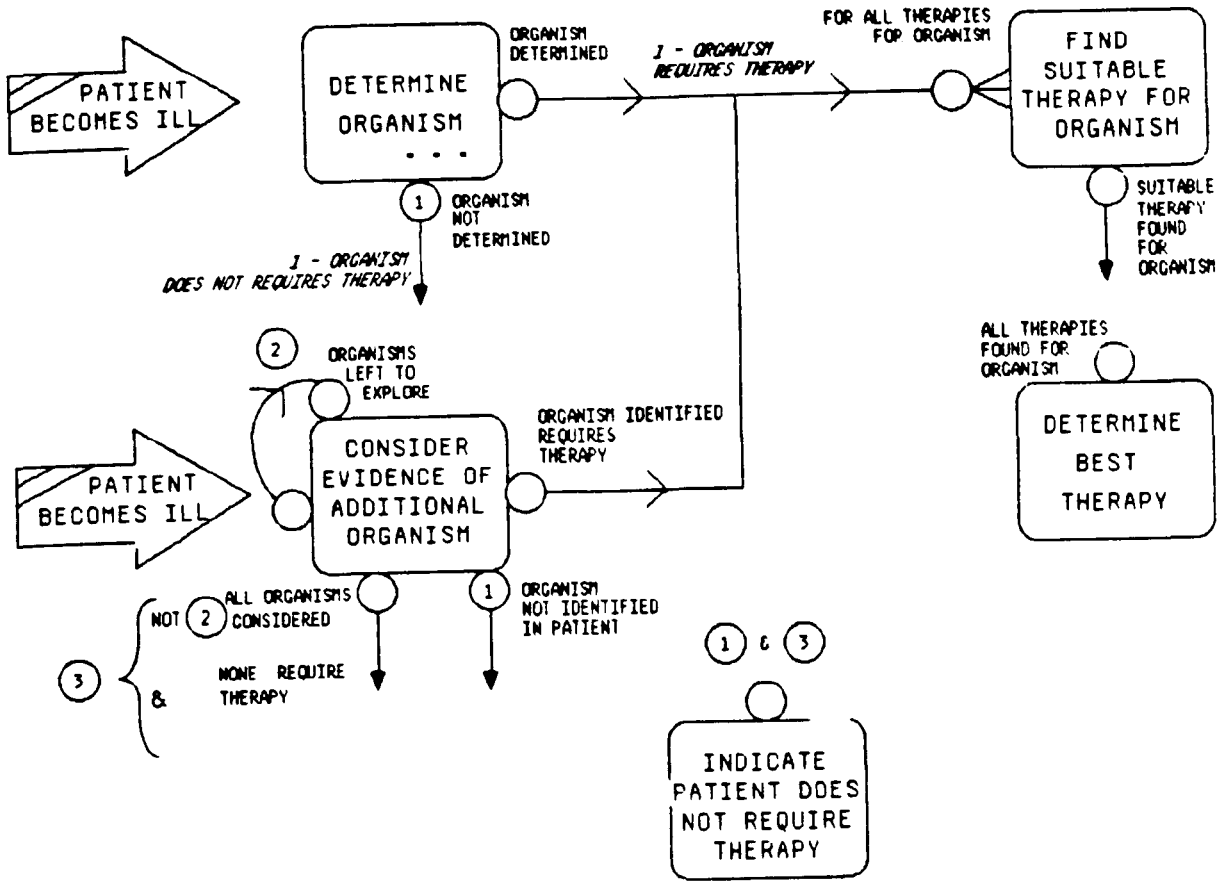
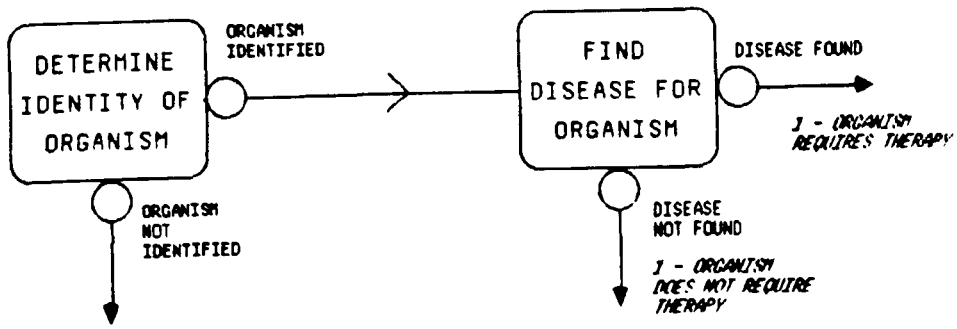


Figure B5.8 Entity Relationship Diagram for MYCIN Rules



DETERMINE THERAPY FOR ORGANISM



DETERMINE ORGANISM

Figure B5.9 Action Dependency Diagram of MYCIN Rules

As this example is of a different nature to the previous two examples, it will be explained in full.

There are two action relationship diagrams in figure B5.9, corresponding to the two rules.

We shall start with the higher-level diagram (uppermost diagram on figure B5.9). First the patient becomes ill. This is indicated by the arrow showing an event. This event is a triggering event and starts the behaviour pattern represented by the diagram. The event triggers two actions, Determine Organism and Consider Evidence of Additional Organism. Determine Organism is decomposed, this being shown on a separate diagram (indicated by the ...). We shall consider this first.

In the determining of an organism, the identity of the organism must first be found. If the organism can be identified then the post-condition of this action is fulfilled (the post-condition logic is the text written beside the optionality circle attached to the action box). If the post-condition is fulfilled then the dependency is followed to the next action, which is Find Disease For Organism. This action establishes if there is an identifiable disease for an organism. It forms part of the premise of the higher-level rule. If the premise is established then the post-condition on the right-edge of the action box is fulfilled and the conclusion can be reached that there is definite evidence (probability of 1.0) that there is an organism

requiring therapy. This is an example of a 'reach conclusion' rule. If a disease is not found (If-Then-Else case) then the post-condition at the bottom of the action box is fulfilled. The ① in the circle indicates that this is condition (1) and enables the same condition to be easily referenced elsewhere.

We shall now consider the higher-level action - Determine Therapy For Patient. We have considered the Determine Organism action, now we shall consider the other action directly triggered by the Patient Becomes Ill event - Consider Evidence of Additional Organism. The content of this action is as its name suggests. There will be a number of rules to achieve it, but we have not included them here in the interests of simplicity. This rule investigates every organism and considers if there is evidence that the patient has it and requires therapy for it. The repitition is shown by recursive application of the rule on every organism. It is depicted by the involuted dependency which is invoked as long as there is an organism still to consider (condition ②). In effect this is 'For All Organisms'. For each recursed occurrence of this action the possibilities are either that the organism exists in the patient and requires therapy, or that it does not exist in the patient (condition ① again), or that it does not require therapy at all. If all organisms have been considered and none of them require therapy then condition ③ is fulfilled - this will only occur once for all invocations of the

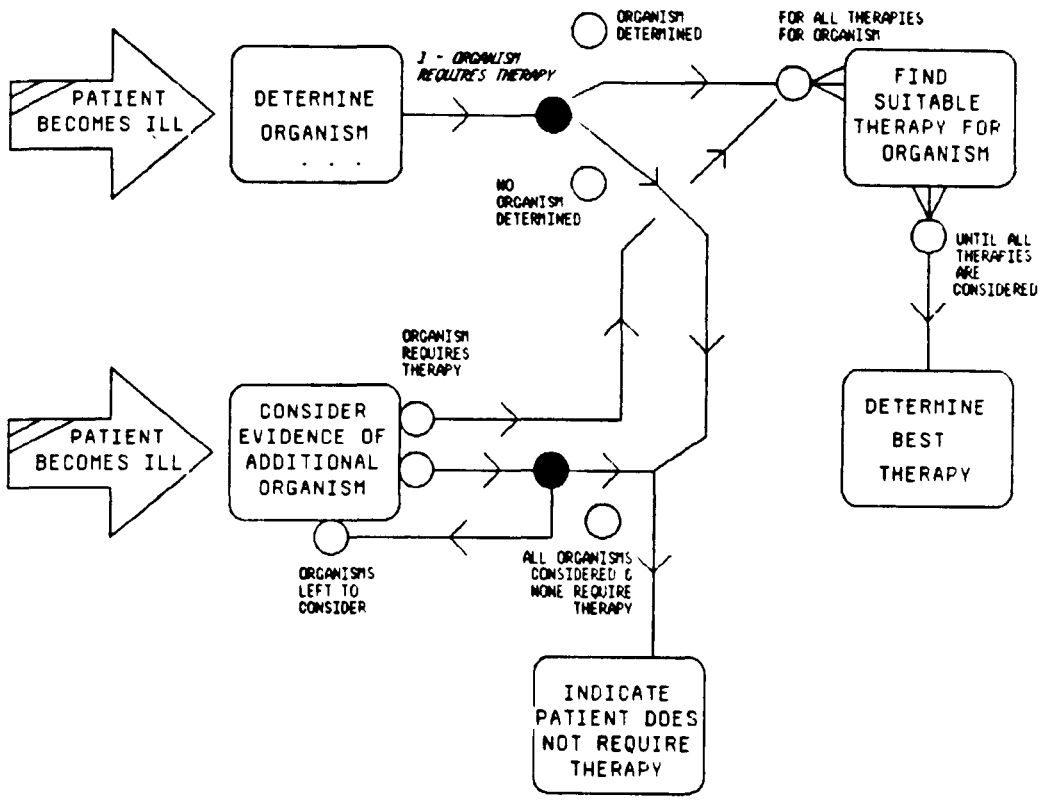
action in the behaviour pattern.

If condition ① and condition ③ are both fulfilled then the 'Indicate ...' action at the bottom of the diagram occurs. This is known because the action has a floating pre-condition (a circle on the edge of the box with no explicit dependency) and the condition logic is ' ① & ③ '. So the instant that condition ① and condition ③ have both been fulfilled, the 'Indicate ...' action occurs and someone/thing is informed that the patient does not require therapy. This is an example of a 'take action' rule; in this case it is the invoker of the behaviour pattern that takes the action.

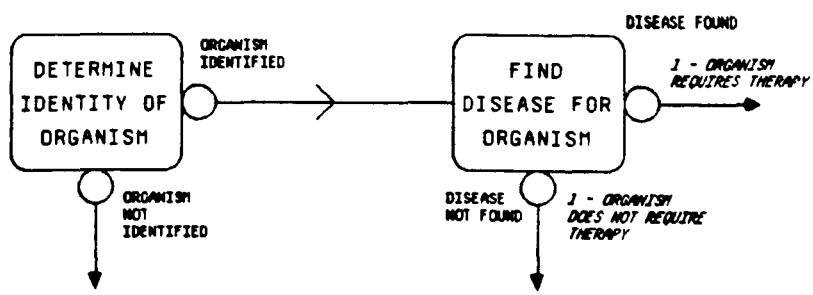
If an organism requiring therapy is found by either of these actions then all suitable therapies for the organism are found. This is shown by the repeated occurrence of Find Suitable Therapy For Organism for every occurrence of the other actions. This action occurs for all therapies for an organism. The result of the occurrences of this action is the compilation of a 'list' of suitable therapies for the organism. Once all the therapies for the organism have been found, the best therapy for the organism will be determined. This is another floating action; it is invoked the instant that all therapies are found for the organism. This action represents a 'take action' rule. The action can either be taken by further rules in this production-rule system, by a separate rule system or even by a human. In any case the action will be based on the list

produced previously. Whatever happens next, the start of this action marks the end of a logical deduction. The taking of the action is a new, separate logical deduction.

The following two figures show the totally procedural and non-procedural specifications of these rules.

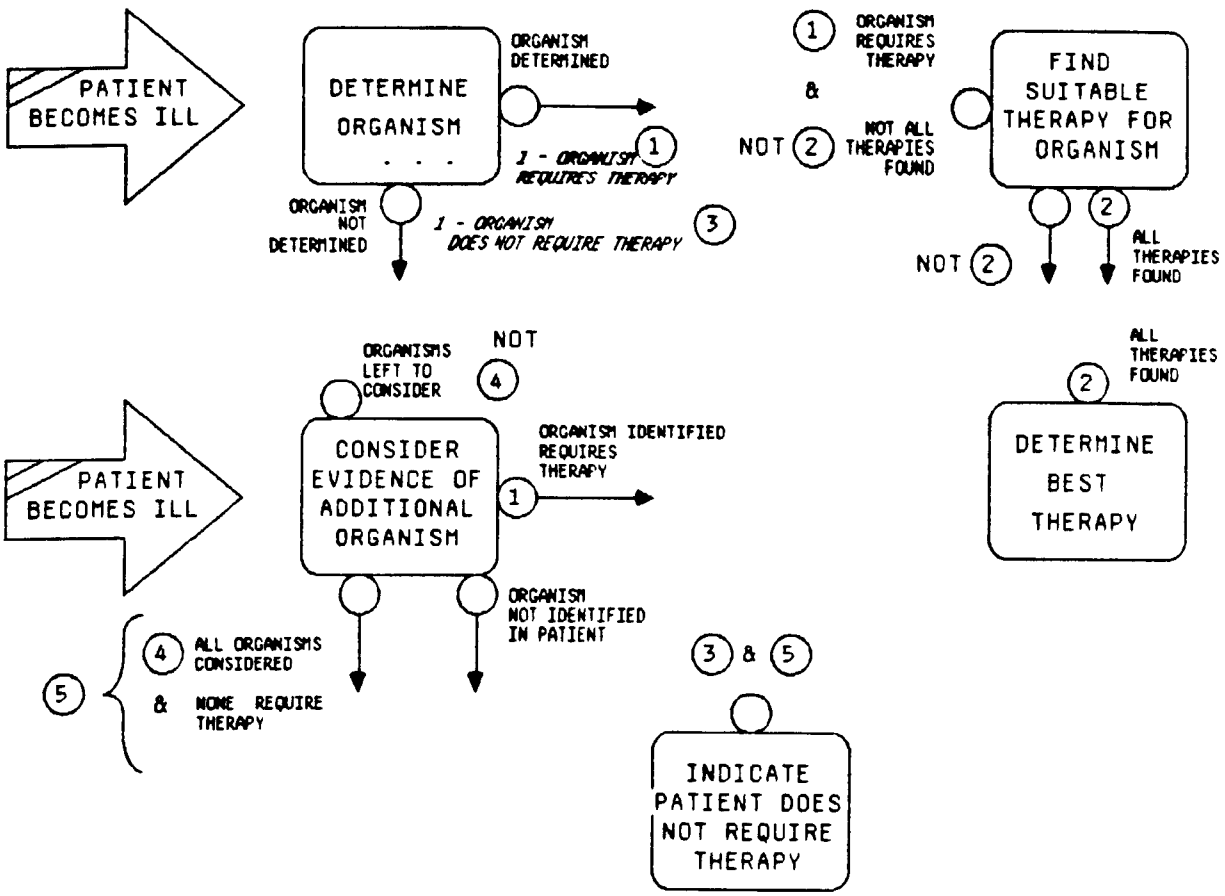


DETERMINE THERAPY FOR ORGANISM

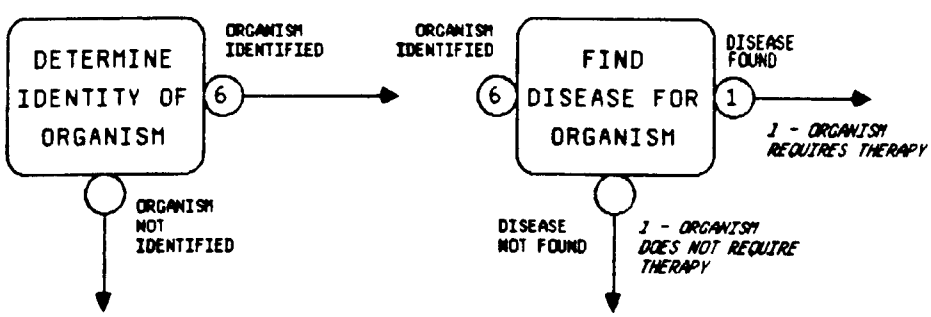


DETERMINE ORGANISM

B5.10 Procedural Diagram of MYCIN Rules



DETERMINE THERAPY FOR ORGANISM



DETERMINE ORGANISM

B5.11 Non-Procedural Diagram of MYCIN Rules

B6 DETAILS OF USE

As of writing, action modelling has only been used once in anger. This was at International Paints, a subsidary of Courtaulds, where it was used on a development of manufacturing systems for their Felling, Tyne & Wear factory. The basic ideas have been assimilated into the Information Engineering methodology.

The main reasons why action modelling has not been used more extensively are a) that it was an unproven technique and analysts are reluctant to be 'guinea pigs', especially at the later stages of a project, and b) that often due to tight timescales elementary processes are not examined in depth anyway; the need for action modelling is vastly reduced in such situations.

B6.1 EXPERIENCE AT INTERNATIONAL PAINT

International Paint were developing a set of manufacturing systems to incorporate Production Planning, Production Scheduling and Work-In-Progress Monitoring. They were using Information Engineering which they had just been taught; I was the JMA consultant involved in ensuring all went well.

The basic analysis went well. An initial analysis was done and the project was split into a number of phases according to the structure of the data and its functional use (see C6 as entity model clustering was used to achieve this split). The phases were Product Planning, Equipment Scheduling, Manpower Scheduling, Work-in-progress Monitoring and Raw-Material Scheduling. The project is not yet complete.

Action modelling was first used in the Production Planning phase and was used in two ways. The Production Planning phase was concerned with the initiation and planning of Factory Orders, being orders on Production for a given amount of a certain paint (usually, though not necessarily).

An entity relationship diagram and a process dependency diagram were developed for the area. Interestingly the process dependency diagram basically covered the life cycle of a Factory Order. However this produced communication problems as there were many lines going into the Cancel Factory Order and Modify Factory Order processes (from almost every other process in fact). In an attempt to overcome this (and in my absence I hasten to add) they more or less turned the dependency diagram

into a data flow diagram with ensuing consistency and specification problems. On my return I introduced them to the concepts of floating activities, which is exactly what the cancel and modify processes were; they act on a Factory Order in almost any state and change it, but are not really affected by where the Factory Order was last changed - the dependencies in this case are really a red herring.

When this was used the dependency diagram became much more communicable. They were able to show the main 'flows' of information through the area unconstrained by what are necessary, but 'side' processes. Floating activities vastly reduced the number of lines on the diagram and enabled the important detail of the diagram to be concentrated on.

Eventually in this phase process logic analysis was required, as they intended to produce a full prototype and needed details of the elementary processes.

By the time process logic analysis was required, the analysts needed a refresher course in the techniques. I showed them process logic analysis, which consists of data access diagrams (a data navigation diagram) backed up by action diagrams (basically pseudo-code). They did not like the access diagrams because of the complexity of the concept and the amount of effort involved; they were even less keen on the action diagrams because, as they said, "we might as well be programming". Under

this criticism I introduced them to the concepts of action modelling, describing it as a detailed form of dependency diagram. They were much happier with this and decided to use it.

I worked with them to produce the first model - a model of a 'Replan Factory Order' process. This was used because it was felt to be the most complex process in the area and enabled them to grasp all the concepts. The analysis proceeded by identifying the 'actions' in isolation and considering their pre-conditions and post-conditions. The explicit dependencies were then simply worked out by matching pre-& post-conditions. The end result can be seen in figure B6.1. I then analysed this process in even greater depth for the purpose of this thesis - they did not need the detail as it was felt that the vast majority of the process would not be automated, if not all of it.

The analysts produced action models for all their elementary processes. I have not been able to include all these diagrams for confidentiality reasons. However the analysts expressed that they felt the models were reasonable and useful; doing the analysis helped the understanding of the individual processes and the complete area. International Paint at Felling has only a single lynchpin Production Planner who is overworked; it was not felt possible to take him away to validate the diagrams, so I have no user response to the technique.

At the time of writing, the project is in design and construction of this phase. No further analysis has been done, so I have yet to see how the technique will be used in later phases.

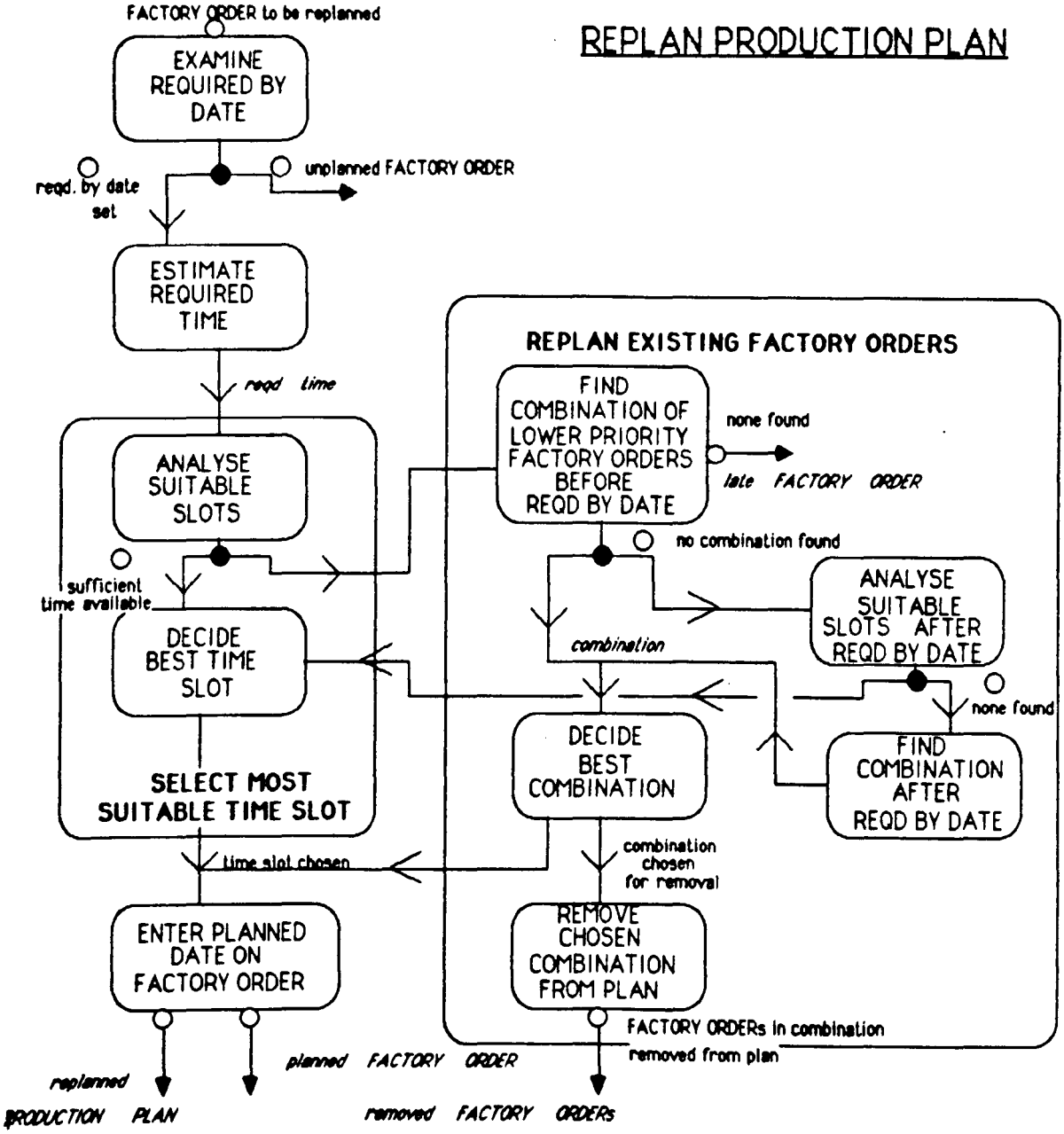


Figure B6.1 Action Model of Production Planning

B7 BENEFITS OF ACTION MODELLING

In B2 I discussed the reasons why action modelling was developed and how action modelling helps in these areas. I should like to point out that the principal benefits of action modelling are in dealing with these problems. As they were considered earlier I will not repeat the discussion. The remaining benefits largely derive from the nature of the technique.

B7.1 IMPROVEMENT TO UNDERSTANDING OF DATA & ITS BEHAVIOUR

In 1974 the Data Dictionary Systems Working Party developed its 'quadrant'diagram [DDSWP, 74]. This diagram, reproduced in figure B7.1, splits into four types - conceptual data (eg. entity types and relationships), conceptual activity (eg. process and function), designed data (eg. a database schema and record types), and designed activity (eg. systems and programs). The mappings between these areas is also required and modelled.

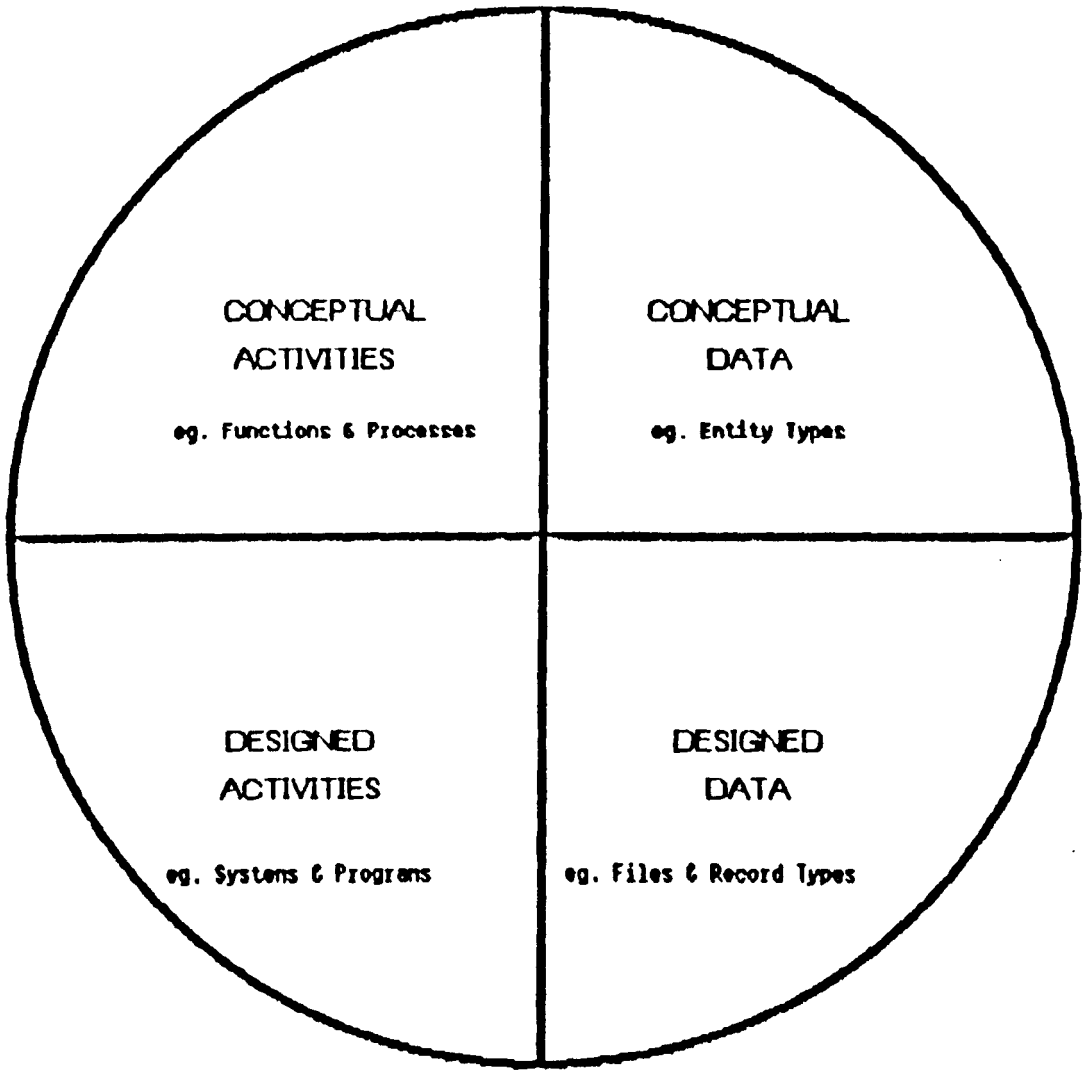
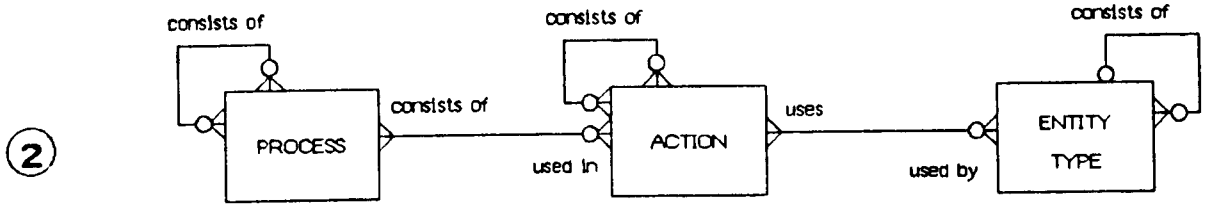


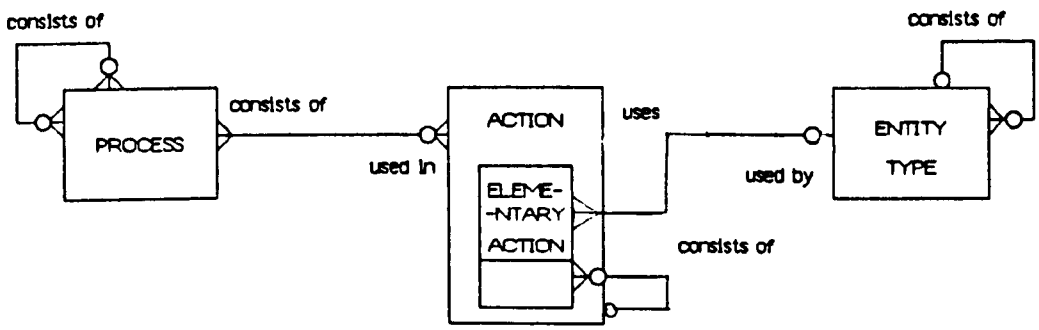
Figure B7.1 DDSWP Quadrant Diagram

But lets look at this mapping in more detail. Ostensibly we have:

If we decompose this many-to -many relationship we find:



Note: If we consider only the relationship between elementary actions and entity types then the situation is:



The many-to-many relationship between Process and Action is due to shared usage of actions as a result of shared usage of data.

② should be modelled in a data dictionary, with actions being considered a 'full-blown' object in their own right. This expands the quadrant diagram into a 'sextant'diagram, see figure B7.2. This consideration of actions as an object was only recognised as a result of action modelling.

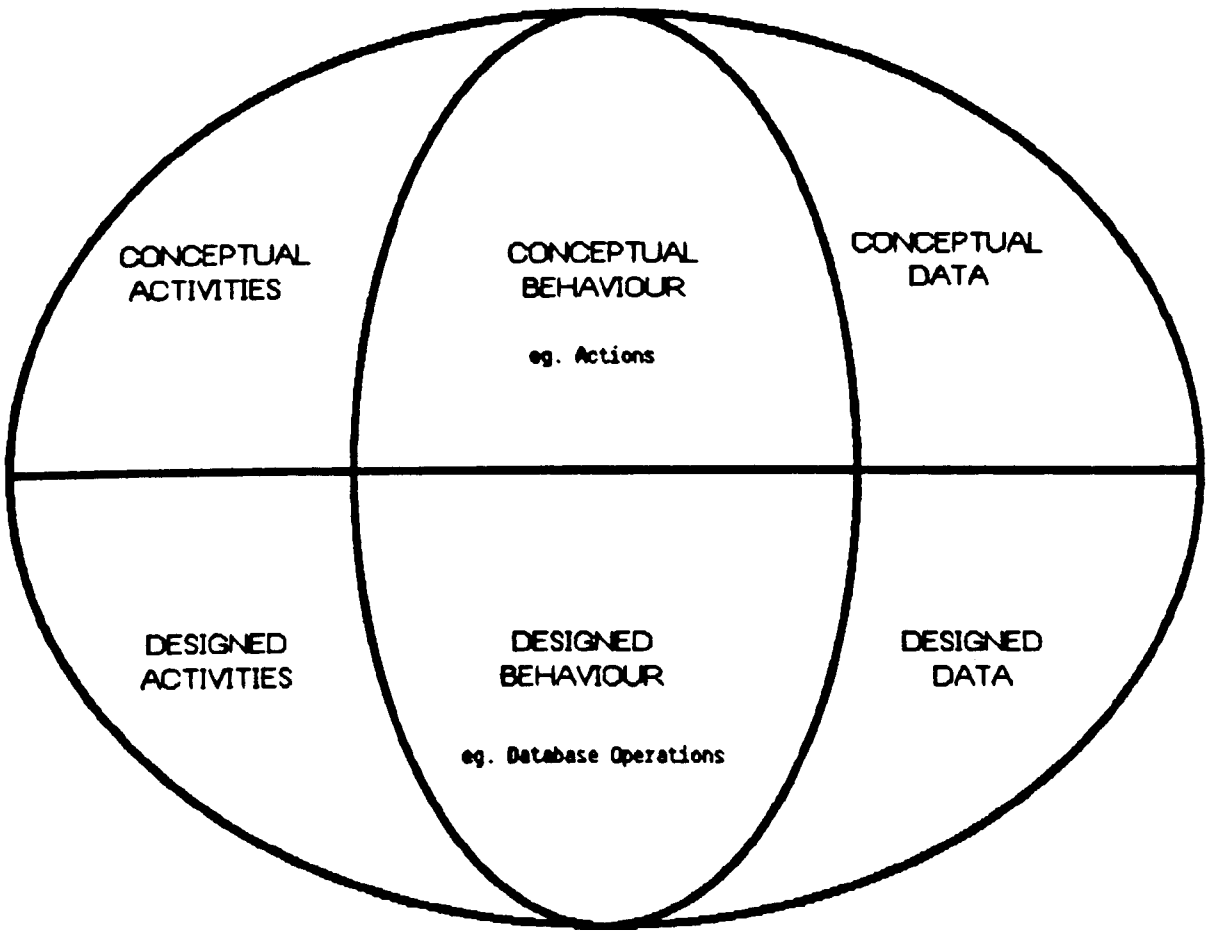


Figure B7.2 Sextant Diagram of Information

B7.2 DIAGRAMMATIC SPECIFICATION AND PROCEDURALITY

There are really two main methods of specifying detailed behaviour, firstly non-diagrammatic techniques, eg. Structured English [deMA,78] and predicate calculus [KOWA,74], (see B8.1.5) and secondly graphics, eg. access path diagrams (see B8.1.2), and entity state diagrams (see B8.1.1). The main benefit of pseudo-logic is its straightforwardness. However there are many drawbacks. The principal drawbacks are a lack of communicability and a distortion of reality. Pseudo-logic specifications are normally one-dimensional due to the nature of the media they are constrained to, eg. they are sequential by default and parallelism is extremely difficult to show. This also applies to the graphical techniques in common usage. As shown by Heitmeyer & Mclean [HEMC, 83], arbitrary decisions forced on analysts by the communication constraints of a media tend to be enshrined by the designer who is not in a position to consider any other possible designs.

There are currently many arguments concerning the benefits of non-procedurality v procedurality. These arguments ignore the point that sometimes you need one, sometimes the other, but most often you need a combination of both. It is probable that there is no such thing as completely procedural or non-procedural data behaviour, a behaviour specification should be procedural where necessary and non-procedural where necessary. It is very difficult to achieve this in textual and pseudo-logical

specifications, again due to the one-dimensional nature of the media they are constrained to.

It is well accepted that diagrams are more communicable that textual specifications, 'a picture speaks 1000 words', and as this thesis is based on that assumption I will not argue with it. However, apart from greater communicability, most graphical techniques for behavioural specification suffer from the same problems as the pseudo-logical specifications. Action modelling on the other hand gives an accurate representation of behaviour and provides a solution to the above problems; it is easily communicable, forces no assumptions as to implementation considerations and allows a flexible choice of procedurality. For example compare figure B5.2, figure B5.3 & figure B5.4. Figure B5.4 is as non-procedural as it is sensible to show, figure B5.3 is as procedural as it is sensible to show and figure B5.2 is a compromise which we think the best. The same comparison can be made between figures B5.9, B5.10 and B5.11. It is important to have the flexibility that an action model allows so that the most intuitive diagram is produced for both the analyst and the person who is the source of knowledge being acquired.

This flexibility allows procedurality issues to be left to designers. If not, designers may be forced to use an unsuitable programming language to enable an analyst's specifications to be accurately designed or, conversely, designers may be forced into

costly, error-prone conversions from one type of procedurality to another.

Non-procedural action models would map well to essentially non-procedural languages such as Prolog; procedual action models would map well to procedural languages such as Pascal; and the 'mix' would map well to languages such as LISP. With action modelling it is very easy to translate from one type of representation to another. Therefore when designing systems from action models, the flexibility allows the designer a free choice of implementation language with no constraints imposed by the representation.

B7.3 INFORMATION FOR DESIGN

Most methods of specifying behavioural requirements result in a lack of information for software design, or strictly confine the design process. The former means that some of the analysis effort will need to be duplicated in the design phase (eg. for the mapping of requirements into an implementation), the latter means that a designer does not have sufficient freedom to produce the 'best design'. Software designers need to be provided with all the information they need for design without a restriction of their choice of acceptable designs.

One of the results of a complete behavioural specification is a definition of the usage of data. This is essential for design

(as opposed to analysis) as it provides a large part of the information needed for that task. In addition a definition of the shared, or common, usage of data is also made available, easing the subsequent modularisation of systems and programs. As there is detail of data usage within processes, action modelling provides a good base for designing the input, output and processing of data. Additionally this knowledge of the usage of data is essential for data structure design.

A complete behavioural specification should enable the design of software to:

- be highly structured
- allow for a high degree of parallelism (as much as necessary)
- be a true representation of any behavioural requirements
- have a high degree of shared code
- have a low degree of repeated code

B7.4 ACTION MODELLING FOR KNOWLEDGE-BASED SYSTEMS

Action modelling was developed to help build basic information systems. It was realised that the information being modelled was very close to that required for knowledge-bases. A small piece of research was done to investigate the use of action modelling for knowledge acquisition. The results were published in [FEFI,85b]. It is interesting to note that the only

knowledge which applies to knowledge-bases and not information systems was the area of premises, conclusions and probabilities (see B3.4.5).

The development of knowledge-base systems, although a much newer discipline than the development of information systems, exhibits similar traits, eg. the real problems are in the knowledge acquisition and representation process rather than in the technical aspects of programming methods. It has been shown that those who have acquired skills and expertise in any task are generally not very effective at communicating the context and process of their mastery. They lose awareness of what they know, they tend to solve problems and make decisions by recognising situations as instances of things with which they are familiar, rather than by the application of general principles and deductive steps that provide causal links between one stage and another of a problem solving sequence [JOHN, 84]. This seems to indicate that the knowledge acquisition process is as difficult, if not more so, in the development of a knowledge-base system as in an information system. So the use of a representational technique developed for information systems, has proven relevant in the area of knowledge-base systems.

We are interested here in two type of knowledge - facts and rules [MICH, 80], [CLAN, 83], [SOWA, 84]. There are many other types of knowledge, for example, inherited knowledge, feelings,

emotions, and so on, which we do not understand well enough to model as yet [KIDD,85].

B7.4.1 Facts

Facts form the underlying static and dynamic data on which knowledge is based [ZELN, 79]. Simple facts generally take the form:

X rel Y

For example:

Paul has Meningitis
Paul is a Patient
Meningitis is a Disease.

From this can be derived the generalisation:

Paul has Disease ①

as Paul and Meningitis are instantiations of Patient and Disease respectively. Statement ① is the classic format of two entity types linked by a relationship, as Patient can be an entity type, Disease can be an entity type and 'has' can be a relationship. Therefore it is possible to represent facts in an entity relationship model.

Instantiations, eg. Paul is a Patient, would not be modelled in an entity relationship model as entity models are concerned with types of data as opposed to instantiations. There are some data models which represent the instantiations, eg. classification in SHM+ - a semantic hierarchy data model [BROD, 83]. Entity types are equivalent to some forms of 'static data' in a knowledge-base, ie. data which stays fairly static throughout uses of the knowledge. Some important instantiations may also appear as facts in a static data base, eg. Meningitis is a Disease. However most instantiations would appear either as dynamic data, ie. be instantiated at the time of use, or appear in rules, eg.

If Patient's name is Paul then Paul is a Patient.

The same situation exists in information systems where certain instantiations are very important to the system, eg. General Motors is a Car Manufacturer in an information system for a General Motors Car Showroom. But most, if not all, instantiations would appear in a database or be coded in tables and programs. Therefore the traditional form of entity relationship model should be as sufficient for modelling facts as it is for modelling data types, and it is very successful at that.

In the modelling of facts, entity relationship modelling

conventions such as optionality and cardinality are all suitable. For example, a Patient may have many Diseases and a Disease may be had by many Patients. Also, facts cannot simply be concerned with the detail of knowledge to be represented, but must cover the whole situation. This is analagous to data representation where data external to an organisation are modelled, eg. Customer and Supplier.

B7.4.2 Rules

Rules tend to have the form:

If premise then ---

[CLAN,83], [ATKIN,83], [CEBR,84], [DAVIS, 80a].

The --- enables the dichotomisation of rules into two main categories. The first category covers rules of the form:

If premise then **reach some conclusion with a probability of its holding**

and represent the case that some conclusion can be reached, albeit tentatively, based on the premise. An example of this type of rule is:

IF [1] the stock's dividend has kept pace with inflation over the past 10 years
AND [2] the stock's dividend has never dropped by more than 10% in any 1 year
THEN there is good evidence (.8) that the dividend is stable

[DAVIS, 80b].

The second category covers rules of the form:

If premise then **take some action**

and represent the case that some action should be taken by someone/thing if the premise holds. An example of this type of rule is:

IF: the most active context is distributing massbus devices
& there is a single port disk drive that has not been assigned to a massbus
& there are no unassigned dual port disk drives
& the number of devices that each massbus should support is known
& there is a massbus that has been assigned at least one disk drive and that should support additional disk drives
& the type of cable needed to connect the disk drive to the previous device on the massbus is known
THEN: Assign the disk drive to the massbus

[McDE, 82].

Premises, conclusions and actions must all be based on, and affect facts about a situation. Premises tend to refer to the state of a situation and /or refer to conclusions of other rules. Thus premises are based on the reasoning of some 'knowledge' and act as 'links' between rules. They can be fairly complex, and often are.

B7.4.3 Action Modelling of Rules

There are two separate uses of actions in knowledge representation. The main use is in the modelling of deducing a premise from which to draw a conclusion or take an action. The second use of actions is in the modelling of any action patterns which need to be taken; this is equivalent to the use of action modelling in information systems.

An action model would be produced for a complete 'production-rule' system, and would consist of a number of linked action dependency diagrams. Action dependency diagrams would be constructed for the two separate uses of actions. Within the first use there would be action dependency diagrams at different levels of abstraction to match the different parts of a deductive reasoning chain. There would probably be a diagram for every conclusion to be reached and a diagram for every action to be taken, though the latter might be represented by separate models, this being the second use of action modelling. Rules are 'linked' by the parts of their premises and the actions involved in ascertaining the truth or otherwise of the parts. The diagrams representing these rules and actions are linked because of this. To summarise, an action model consists of a hierarchy of diagrams with each lower-level diagram representing rules which are concerned with evaluating a situation for rules represented in higher-level diagrams.

Section B5.3 gives an example of the use of action modelling in this way.

B7.5 FINDING ELEMENTARY PROCESSES

One of the thorniest problems in Function Analysis is knowing when an elementary process is reached, where elementary process is defined as the lowest-level of activity that is 'logically complete'. Logically complete here means that:

* the process is not dependent on any previous processes except that they manipulate the enterprise's state for the process to occur
* at completion any violation of state integrity have been made good
* the process is not dependent on any subsequent process to make good state integrity violations

For example, Update Bank Balance to be elementary must leave the enterprise in a consistent state, must have fully followed bank policy on balance updating (eg. application of bank charges), must not be dependent on previous processing except that the state indicates that the balance requires to be updated, and any subsequent processing must assume that the updating of the balance and any necessary attendant processing has been completed.

A benefit of action modelling is that the insight it generated enabled this more stringent definition of elementary process. Also, based on this definition, action modelling can be used to identify whether a process is elementary or not.

Based on the above definition, an alternative definition can be given:

> An elementary process is a collection of dependent actions where no action in the collection is dependent on any actions external to the collection (ie. has optional dependencies leading into and out of the collection), but where every action internal to the collection is dependent on at least one other action in the collection.

The justification for this is:

* the independence of Processes to be elementary implies that there must only be optional dependencies between associated processes; therefore, the terminating action of one process must be optionally dependent on the commencing action of a related process,

* state integrity demands that an action which violates a state rule must be mandatorily related to the action(s) which restores the integrity.

So to find an elementary process just examine an action model of the area and isolate a collection of actions which meet the definition. Note that a totally procedural action model will be needed for this.

B7.6 TEMPORAL MODELLING

There are two types of temporal modelling **time modelling** and **process modelling**. "Research in time modelling is concerned with the representation of the times of events and of timing relationships between events. The major objective of this

research is the mechanisation of inference about these times and timing relationships. Research in process modelling is concerned with the representation of the internal structure of events and processes, and with how events and processes affect the state of the world" [BODE, 83].

Most time modelling research leads to the introduction of techniques to handle modelling and inferencing about actual timing information of data and its processing, eg. by the addition of 'time stamps'. Some research is concerned just with implementing database timing mechanisms and some with a spectrum from conceptual modelling through to implementation. This area is very heavily researched; the majority of the results seem to be concerned with the implementation of a method of recording and enquiring upon the times when something occurred. Time modelling is considered to be outside the scope of this thesis.

Process modelling basically deals with the dynamics of a system, ie. modelling its behaviour, as opposed to data modelling which deals with statics. Action modelling can plainly be seen to lie in the realm of process modelling. So action modelling is a method of temporal modelling, which it does by modelling system dynamics.

B7.7 SUBSTITUTE FOR OTHER BEHAVIOURAL MODELLING TECHNIQUES

As will be discussed in B8, action modelling deals with the same information as other behaviour modelling techniques, eg. entity state diagrams and data flow diagrams. It is possible to use action modelling instead of these techniques, if it were desired. The benefit would be a reduction in the number of different techniques that neeeded to be used within an analysis project. The disadvantage to this is that the substituted techniques tend to be more expressive in their particular domains than action modelling, ie. action modelling is a general modelling technique. For example, an action model could be constructed of all the actions applying to a given entity type, but an entity state diagram is probably better at doing this.

B8 COMPARISON OF ACTION MODELLING WITH OTHER BEHAVIOUR MODELLING TECHNIQUES

Some techniques are already in use for behaviour modelling. However these do not meet the needs identified in B2. This section compares action modelling to some of these other techniques.

Action modelling is unique in a number of aspects, for example it is the only user requirements specification method dealing with detailed behaviour and it is the only method with such a close symmetry with data. However there are also a number of aspects where action modelling is not unique; these are mainly concerned with considering aspects of depicting the actual elements of processing, for example, actions themselves. The following deals with the original and the less-original aspects of action modelling in comparison to some of the existing detailed behaviour modelling techniques. The majority of techniques in this comparison are diagrammatic in character. Non-diagrammatic methods are dealt with together (B8.1.5).

The comparison appears in three parts, comparison to individual techniques, comparison to parts of particular methodologies which make use of a particular behaviour modelling technique and thirdly, details of the behaviour modelling techniques of certain high-profile system development methodologies. The following is not intended to be exhaustive, just to deal with techniques which might be considered to have common

characteristics.

There are various conclusions which can be drawn from the following comparison. The most unique aspect of action modelling is its use as a Business Area Analysis technique rather than a Business Systems Analysis technique (see X1.5), ie. it can be validly used in a different part of the system development life-cycle to other behaviour modelling techniques; there are other behaviour modelling techniques for modelling inherent logic of processes but they model design rather than requirements. An important difference is that action modelling has a close symmetry with data which is very beneficial (see chapter D); most other techniques do not exhibit this. Other major differences to the majority of the techniques described below are the use of abstraction on a single model and a comprehensive modelling of data behaviour, especially of the interactions between behavioural components. There are few actual original components of action modelling, its main originality lies in the unique combination of components and the use intended to be made of them.

B8.1 TECHNIQUES

B8.1.1 Entity Life Histories/Entity State Diagrams

Entity life histories [JACK, 83], [HALL, 82] and entity state diagrams [JMA,86b], [ROEV, 81] are used in modelling all of the

major state changing transactions on a particular entity type. Not all transactions on an entity type will be modelled, only those which have a significant effect on that entity. Thus life history state diagrams model the significant behaviour of single data types. While this could be modelled in action modelling, the life history and state diagrams are probably more expressive in this matter; however they do require user education into their different conventions. Action models are capable of modelling much more than life history and state diagrams, for instance, action models model the interactions between different entity types and the conditions under which behaviour occurs.

Entity life histories have the same sort of detailed modelling concepts as action models, but have fewer. For example, they model the sequence, iteration and selection of events which can occur on an entity type. These concepts are also action modelling concepts, but this is the totality of an entity life history, whereas it is just a basis for detail in action modelling. Therefore action modelling provides a more comprehensive set of conventions.

B8.1.2 Access Path Diagrams/Process Logic Diagrams

Access path diagrams [GASA, 79], alias function logic diagrams [MACP, 82], a.k.a.logical access maps [MAFI, 83] and process logic diagrams [JMA,86b], model the way that a given process will make use of entity types and its relationships. This is a

similar area of modelling to action modelling. Neither have to be done just for a single process, but commonly are; however action modelling provides information to discern whether a single process is actually being dealt with or not.

The most striking difference between the two is that action modelling provides a method of recording user requirements separate from any design preconceptions or need to provide design information, whereas access path diagrams record a level of detail sufficient for programming and require user discussion as to whether data is created, deleted, modified etc., and which database access paths will be used (masqueraded as which relationships are used). As well as dealing with a different level, action modelling also has the ability to record abstraction, an ability not possessed by access path diagrams. Additionally, access path diagrams often turn out to be difficult for inexperienced people to use and understand due to placing functional information on an entity relationship diagram. Such diagrams are tedious to produce and are totally uncommunicable for any complex process (there is little point using them for simple processes).

B8.1.3 Data/Information Flow Diagrams

Data flow diagrams [GASA, 79], [deMA, 78] are very widely used to model the functional transactions in a modelled area and the transactional flow of major and/or necessary information.

They are most often used within structured systems analysis methods, consequently it is often difficult to separate the data flow diagram technique from surrounding methodology. In this section we are dealing solely with the diagrams themselves, unless otherwise stated. Data flow diagrams have many failings [BIBR, 84], but have one saving grace being user intuitive, hence useful for communication. Data flow diagrams are used at a different level of analysis to action modelling as they capture high-level functional requirements. They are used in Information Engineering, but only to describe existing or designed systems.

I have not found a proposal to use the diagrams themselves for capturing detailed processing requirements, probably due to the vast amounts of documentation this would generate with every new level of abstraction having to appear as a separate diagram. Furthermore, data flow diagrams have a paucity of facilities for capturing detail, eg. dependency, cardinality and conditionality. Often data flow diagrams are associated with methodologies which use other techniques for capturing this detail, eg. pseudo-code.

Information Engineering and D2S2 [MACP, 82] have dependency diagrams, which are basically data flow diagrams without data flow but with the detail. These also have never been proposed for detailed requirements gathering. Thus action modelling is

different to these techniques in the level of processing considered the detail captured, and by the use made of abstraction to enable the detail to be managed and comprehended.

B8.1.4 Petri Nets

"A Petri net is an abstract, formal model of Information Flow .... The major use of Petri nets has been the modelling of systems of events in which it is possible for some events to occur concurrently but there are constraints on the concurrence, precedence or frequency of these occurrences" [PETER, 77]. Petri nets consist of two types of node, places and transitions, connected by directed arcs from either place to transition or vice-versa. A node cannot be directly connected to a node of the same type, ie. places can only be connected to transitions and vice-versa. Petri nets are a very detailed tool, "for practical purposes a descriptional tool on a higher level than Petri nets is required" [RIDU, 82]. Richter & Durchholz use two Petri net derivatives for system development, "Channel/agency-nets which are suitable for information flow overviews without sequencing and concurrency specifications" and "Predicate/transition-nets which allow for formal specification of any desired degree of precision and detail".

Another proposed use of a Petri net derivative is Sakai [SAKAI, 83] who has closely integrated a Petri net specification with the entity relationship approach to model the behaviour of data.

The above two methods characterise the methods of behaviour modelling using Petri nets. The Sakai type keeps very much to the initial definition of Petri nets. Peterson [PETER, 77] shows how Petri nets can be used as an equivalent to finite-state machines. The Sakai type uses this idea to model the changing states of entity types and relationships in an equivalent to entity state diagrams (see B8.1.1), and with the same advantages and disadvantages as entity state diagrams.

The Richter & Durchholz type (henceforth refered to as R&D nets) have extended Petri nets by expanding transitions into complete functional descriptions. The overall effect is similar to a data flow diagram with data stores between every function. However, in my opinion, the effect actually achieved is confusing and, hence, less useful for communication than data flow diagrams.

If the Petri net definition at the beginning of this sub-section is examined, it can be seen that Petri nets are intended to cover broadly the same type of modelling as action modelling. The application of Petri nets à la Richter & Durchholz bring their utilisation even closer to that of action modelling. Thus R&D nets model similar concepts to action modelling and for similar purposes.

The various similarities and differences between action

modelling and R&D mets will now be considered in detail. The R&D net equivalent to the action concept is an expanded Petri net transition node. This node represents the functional transition from one data state (or set of data states) to another, ie. an action. The main difference between them is the close relationship between an action model and a data model; there are rules governing this in action modelling but nothing similar in R & D nets. Another difference is the use of abstraction, which can be shown on a single model in action modelling. In R&D nets abstraction is dealt with either by having a separate lower-level model (cf. data flow diagrams) or by having a separate pseudo-code definition if there is insufficient material to warrant a lower-level model.

In R&D nets there are no associations between the functional transitions per se as every transition must be connected to a 'data store' place. Associations between transitions can be elicited by examining the relevant edges from transitions to places and vice-versa. There is no R&D net equivalent to dependency or conditionality separate from a functional description; both concepts are of great importance to action modelling. Furthermore there is no R&D net method of modelling the pre- or post-conditions which are needed for behaviour modelling. To summarise, although R&D nets can model the same area as action modelling, it is not complete in the area of action interaction, this being one of the areas with the most emphasis in action modelling.

B8.1.5 Non-Diagrammatic Techniques

It is possible to use Structured language to describe behaviour, for example Structured English [deMA, 78] and action diagrams [MART, 85a]. The problems with this were discussed in B7.2. The two main considerations are lack of communicability and inadvertently forcing design issues too early. Structured languages can describe anything described by any other technique, as they have the power of natural language behind them. However we know diagrams are better, so we should use them.

B8.2 METHODOLOGY BASED TECHNIQUES

Any complete methodology must have a method of capturing information about detailed behaviour. Most methodologies stick to the tried and tested but user unfriendly method of some form of pseudo-code (see B8.1.5). This section deals with some methodologies which have developed some unique diagrammatic technique to cover this area instead of, or in addition to, pseudo-code.

B8.2.1 ACM/PCM - Active and Passive Component Modelling [BRSI, 82]

As the name suggests, ACM/PCM consists of modelling the active parts of a system (behaviour modelling) and the passive parts of a system (data modelling). Here we are only concerned with

'ACM'. ACM has three major concepts, database operations, actions and transactions; transactions use actions, which use database operations. Actions and transactions both have pre- and post-conditions and their formation is guided by the data model. As can be seen, this is a similar set of concepts to action modelling. ACM models its behaviour by taking a part of the data model and drawing arrows to represent the actions and database operations required, this being similar to access path diagrams (see B8.1.2).

There are only 2 levels of behaviour modelling in ACM, behaviour schemes (described above) and a pseudo-code specification in a language known as BETA. Thus action modelling and ACM differ in how behaviour is modelled and in the modelling of interaction, action modelling models interactions, ACM doesn't.

Concerning behaviour modelling, action modelling has a complete, consistent set of diagrammatic modelling conventions enabling a picture to be built up of behavioural requirements; ACM relies on the data model, which can result in a loss of expressivemess and an increase in confusion for a user.

Concerning interaction modelling, because ACM uses the data model it would be difficult and contrived to model the interactions between actions and between database operations. In action modelling this is natural and powerful but, above all, necessary for a complete behaviour specification.

Another difference to notice is that ACM discusses database operations making this part essentially a Business System Analysis or even a Business System Design technique.

B8.2.2 Remora - Richard and Rolland [RIRO, 82]

Remora has two steps, a conceptual development step and a logical development step. The conceptual step "is a formal representation of the natural structure of facts perceived in their static and dynamic dimensions" [RIRO, 82]. This is achieved through two 'schemas', a 'data schema' representing the organisation components and a 'dynamics schema' representing the organisation behaviour. One of the basics of Remora is an expression of any information gathered through a 'formal language' and by a modelling of theoretical concepts and methodological rules.

In Remora "the dynamic dimension is completely represented by three categories of associations between three categories of phenomena" [RIRO, 82] (the phenomena being objects, events and operations). The three associations are the operational modification of an object, the triggering of an event by an object and the 'ascertaining' of object state changes as events. A Remora event is basically the same as an action modelling event. A Remora operation "is an action that can be executed at a given time in the organisation and that modifies

the state of one or more objects" [RIRO, 82]. The Remora dynamics sub-schema is a diagram which "represents the organisation behaviour rules through the interrelations between ..." objects, operations and events.

So the dynamic sub-schema attempts to model the same conceptual items as action modelling. However the result is vastly different. Associations between operations are achieved through the ascertaining of events on objects and vice-versa, with operations being the means of achieving effects and having little importance per se. Action modelling takes completely the opposite view with the operation/action and its affect on data having most importance; external events are considered as the means of triggering the 'effects' and internal events are rarely needed, if ever. Further to this, dynamic sub-schemas have a lack of abstraction and detail about operation interactions. The diagrams have little actual detail but still deal at a low-level of processing; this is probably the worst combination. Additionally, dynamic sub-schemas are incomplete without the accompanying 'formal language'specification.

To summarise, both the intention and use of dynamic sub-schemas are different to action modelling. Dynamic sub-schemas are based on low-level event/object modelling, while action modelling is based on 'medium-level' action-object modelling. So, although Remora models the same conceptual area as action modelling, it takes a different approach to do so.

B8.2.3 Information Engineering, Before Action Modelling

Information Engineering's detailed behaviour modelling techniques are process logic diagram (a form of access path diagram - see B8.1.2) and action diagrams [MART, 85a] (a form of structured language). These have all been discussed previously in relation to action modelling. Action modelling was developed specifically to fill the perceived gap between decomposition and dependency diagrams and these techniques.

B8.3 BEHAVIOUR MODELLING IN OTHER METHODOLOGIES

The above discussed various methodologies and techniques for behaviour modelling. This section considers in overview how some other high-profile methodologies deal with behaviour modelling. No comparison is attempted to action modelling as the majority of techniques discussed below were considered in B8.1.

B8.3.1 LSDM - LBMS Structured Development Method and SSADM - Structured Systems Analysis and Design Method [HALL,82], [BURC,85]

LSDM and SSADM have the same base method. LSDM is the version marketed by LBMS, a computer consultancy, and SSADM is the version promoted in the British Government by its CCTA. For processing analysis LSDM/SSADM uses a combination of data flow diagram and entity life histories, plus LSDM/SSADM propose

Structured English for detailed functional design.

B8.3.2 JSD - Jackson System Development [JACK,83]

In JSD, entity life histories are built for every 'entity' using the conventions of Jackson Structured Programming [JACK, 75], this being the origin of entity life histories anyway. Other diagrams are built for operations which are needed to back up the entity life histories; these also use Jackson Structured Programming conventions. JSD has 'process' diagrams which show the input and output of Jackson 'functions' in a similar form to the R&D nets described above (see B8.1.4). The detailed specification of a function is achieved through a form of pseudo-code.

B8.3.3 ISAC [LUND, 82]

"In the ISAC approach (to systems development), information systems are specified on three levels:

1 Change Analysis
2 Activity Studies
3 Information Analysis" [LUND, 82].

These specifications are based around three diagrammatic techniques: A-graphs, I-graphs and C-graphs. A-graphs consist of the high-level activities and the things which are input to

and result from these activities. Thus A-graphs are a form of data flow diagram. I-graphs and C-graphs are different methods of information modelling rather than behaviour modelling.

B8.3.4 CIAM - Conceptual Information Analysis Methodology [GKB, 82]

CIAM is loosely based on the 'infological' approach described by Langefors [LANG, 66]. CIAM attempts to build a 'declarative' conceptual model with a heavy temporal basis. Behaviour modelling in CIAM is complex with a number of associated, detailed methods used. Every entity type has existence criteria which describe some of its behaviour. CIAM also has events and 'relationship functions' which model the behaviour of associated entity types as well. Finally there is the possibility of describing detailed 'information requirements'. All of these concepts are described through a pseudo-code representation. CIAM's behaviour modelling is distributed around the concepts which it affects rather than having fewer, more comprehensive descriptions as is usual. This distribution is probably largely due to the different philosophy of CIAM as compared to the other methodologies considered here.

B8.3.5 NIAM - Nijssen's Information Analysis Method [VEBE,82]

NIAM is a method of analysis, normally for information systems. As in Information Engineering, NIAM's main emphasis is information modelling, with the functional side 'tacked on' for

method completeness. NIAM's functional modelling consists of information flow diagrams, a form of data flow diagram which is restricted to a particular functional area, functional decomposition to form a hierarchy, and conceptual grammar language, which is a particularly procedural form of pseudo-code.

B8.3.6 Structured Systems Analysis (SSA) - de Marco [deMA, 78] Gane and Sarson [GASA, 79]

SSA methods are heavily based around data flow diagrams, almost to the exclusion of anything else. Some, for example Gane & Sarson, make use of other behaviour-oriented techniques to back up the data flow diagrams, for example access path diagrams, and de Marco proposes Structured English.

B8.3.7 D2S2 - Macdonald and Palmer [MACP, 82]

Information Engineering is largely derived from D2S2, especially in the area of behaviour modelling, so all the points discussed earlier in B8.2.3 apply to D2S2 as well.

B9 FURTHER RESEARCH IN ACTION MODELLING

There are some parts of action modelling which were identified as being worthy of further research, but not during this project - usually because of the length of time it would take to do it properly.

B9.1 ACTION NORMALISATION

Codd's theory of normalisation [CODD, 70] has had great impact on data modelling. It was felt that it would be interesting to investigate if a similar property applied to activity, eg. to separate out repeating activity, partially dependent activity and transitive activity into separate actions. On the surface there seems to be no problem with this; the consequences and implications of this are unforeseeable without further research.

B9.2 ACTIONS AND DATAFLOW ARCHITECTURES

Part of the 'fifth generation' computing activity is the development of dataflow architectures - that is highly parallel processes whose operation is dependent on 'dataflow'.

Brief research indicates that there is a relationship between the ideas behind dataflow architectures and those behind action modelling. This is not surprising as they share a common base in parts. The main similarity is in the triggering of actions. Actions occur when all necessary previous actions have occurred and all data has arrived; this is the same as the processing of an instruction in a dataflow machine.

Again the ramifications of this are difficult to tell off-hand; one is that it should be possible to model for 'fifth generation' architectures with action modelling, I don't believe fifth generation architecture modelling is possible without it or something similar.

B9.3 ACTION MODELLING AND DECISION SUPPORT SYSTEMS

As described in B7.4, action modelling can be used to represent knowledge for an expert system. As expert systems are one of the six types of decision support system [ISA, 84], it would be interesting to investigate if action models apply to the other five types. This could be an extensive research project.

B9.4 AUTOMATIC GENERATION OF SOFTWARE FROM ACTION MODELS

An action model describes the inherent logic of a process at a reasonably detailed level. Ideally it would be possible to generate software directly from an action model with no design work or more detailed specification. To achieve this it is likely that a 'programming' expert system would be needed. Additionally a knowledge of the business rules and policy to be embodied in the software would be needed. The sort of programming knowledge required is something like Jackson Structured Programming [JACK, 75]. Knowledge is also needed to know how to translate an action's input into its output, something which people do with occassional difficulty. However it should be achievable.

CHAPTER C
ENTITY MODEL CLUSTERING

CHAPTER C ENTITY MODEL CLUSTERING

C1 INTRODUCTION TO THIS CHAPTER

In this chapter I first discuss why entity model clustering is needed, ie. the reasons why entity model clustering was developed (C2). I then consider the concepts which go to make up clustered entity models (C3) and how to do entity model clustering (C4). An example of the entity model clustering process is given (C5), and this is followed by details of its use in commercial situations - as far as confidentiality will allow (C6). C7 gives a number of benefits of entity model clustering and clustered entity models and C8 compares the technique to other similar techniques. Finally C9 gives details of further research areas identified.

Entity model clustering was developed in collaboration with Whitbread & Co Plc to address a recognised problem where there was no available solution. It is primarily an aid to the communication of large entity relationship diagrams. The technique is based on the abstraction of a conventional diagram to form a linked hierarchy of diagrams and a largely algorithmic method was described to achieve this. However, human direction

and intuition are still necessary due to the nature of the models which form the input to the method and due to the need to make the models communicable to humans.

The technique does not alter the basic information content of an entity relationship diagram as there is no loss of information in the clustering process. If anything the opposite is true, because the technique allows information to be gleaned which is not easily extractable from conventional models.

The technique has been used in both the Information Strategy Planning and the Business Area Analysis stages of Information Engineering. The main originality of the technique is in the controlled structuring of a model to improve its communication and maintenance properties. As described in C8, until very recently I have not found another technique for dealing with this area, and I have still only found one similar technique. The benefits which ensue from using entity model clustering are powerful enough to prove its worth. But suffice to say that the technique is firmly established in Information Engineering and has been used in some of Britain's largest organisations.

C2 WHY ENTITY MODEL CLUSTERING?

C2.1 THE PROBLEM

It is widely proclaimed that diagrams convey more meaning than either textual specifications ("a picture speaks a thousand words"), or data dictionary output, this being essentially 'structured' text anyway. However the usefulness of any diagram is inversely proportional to the size of the model depicted. I reckon that any diagram with more than about 30 entity types is reaching the limits of easy comprehension, depending on the number of relationships; the more relationships the less comprehension is possible due to the accompanying increase in complexity.

Large diagrams tend to be spread over very large pieces of paper with many long (possible tortuous) relationships, many crossing lines and the introduction of many connectors. An example is where diagrams are displayed on walls, with the problems that may ensue in transportation, copying and presentation. The net result is a diagram which is very difficult to understand, present and reproduce. This, combined with the difficulty of implementing necessary changes introduces an element of undesirable instability into models which are otherwise very valuable. Diagrams are often artifically constrained to the display medium in an attempt to overcome the problems of large diagrams, but this has a consequential loss of valuable detail;

for example, overview diagrams are often constructed to fit on a sheet of paper by ignoring a lot of important details. The effect of all these complications is to reduce the value of entity relationship diagrams.

These inadequacies suggest that entity relationship diagrams are not used to their full potential in non-trivial situations. What is needed is a method that allows entity relationship models to be usefully applied on a large scale in such a way that representations of them are easy to maintain, readily comprehensible, stable and yet provide adequate detail for development and planning.

C2.2 THE SOLUTION

Due to these problems, a technique is needed to enable entity relationship models to be applied on a large scale with no erosion of their usefulness. Entity model clustering is such a technique, and was developed specifically to overcome the problem. The result of applying entity model clustering is a clustered entity model.

Clustered entity models are easy to maintain and comprehend, while being resilient to change and provide as much, or little detail as required. The technique has its foundations in tried and tested structuring techniques combined with a full set of diagramming conventions (see C3). When applied to an entity

relationship model the technique allows entity types to be viewed at various levels of complexity, independent of organisational constraints, but reflecting the business context. The structure is based on the relationships of the contained entity types, as opposed to the way the entity types are used or perceived. However in as much as use and perception actually affect the relationships shown between entity types, use and perception are reflected in a clustered entity model.

C3 ENTITY MODEL CLUSTERING CONCEPTS

C3.1 MODELS AND DIAGRAMS

Entity relationship diagrams are a method of diagrammatically representing an entity relationship model. For the following it is important to understand the difference between a model and its representation. For example, a photograph of a model of a bridge is not the same as the model of the bridge (and neither are the bridge itself). The photograph can be enlarged, reduced, added to another photograph, and so on. However, no matter how the photograph is manipulated, the model doesn't change, nor does its meaning. The photograph is a representation of the model which can be changed to enable the model to be more easily comprehended, without changing the meaning or content of the model. (Manipulating the model or its representation does not affect the bridge one whit.) As with the photograph, entity relationship diagrams can be manipulatated in various ways to enable the underlying entity relationship model to be more easily comprehended.

An entity relationship model consists of entity relationship diagrams representing the model, plus definitions and descriptions of all the objects in the model. Typically the definitions and descriptions are kept in a data dictionary or system encyclopaedia.

C3.2 CLUSTERED ENTITY MODEL

Basically, a clustered entity model is a hierarchy of successively more detailed entity relationship diagrams with a lower-level diagram appearing as a single entity type on the next higher-level diagram (cf. structured data flow diagrams where activities are decomposed into more detailed self-contained diagrams [GASA, 79]). This results in a single model consisting of a 'tree' of entity relationship diagrams.

So a clustered entity model is a 'linked' set of entity relationship diagrams plus definitions and descriptions of its objects. A clustered entity model has the same content as an unclustered model, it is just structured in the clustered model.

The basic constituents of a clustered entity model are:

* Major Entity Types (see C3.3)
* Subject Areas (see C3.4)
* Entity Relationship diagrams of every subject area and their definitions and descriptions

C3.3 MAJOR ENTITY TYPES AND MINOR ENTITY TYPES

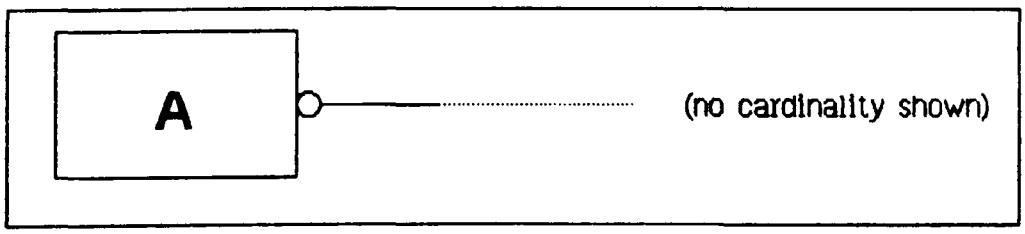
Experience showed that it was desirable to duplicate certain entity types in a number of branches of a clustered model; examples of these are Product, Supplier and Customer. These duplicated entity types were found to be fundamental to a modelled organisation and hence, have been termed **major** entity types. As a broad generalisation, major entity types tend to relate to data which would appear as 'master' files in batch computer systems (also known as 'standing' data). A major entity type is one which is both fundamental and which affects a number of parts of an organisation (ie. is shared).

Entity types which are not major are called 'minor'; these tend to relate to batch 'transaction' file data. Both major entity types and minor entity types should be about the same level as Third Normal Form relations, see [CODD, 70]. (This is the same in both clustered entity models and unclustered models.)

The major entity type concept is a novel one. However entity types with this property were recognised before, just not formally. None of the previous similar concepts were as comprehensive or complete as the concept of major entity type.

For example, Rosenquist [ROSE, 82] defined a concept of 'Principal' entity type which is either: "(a) an entity (type) upon which the existence of other entity (type)s in the

information systems universe depends, or (b) an entity (type) which can exist autonomously". The problem with this is that it encompasses too many entity types, as it just considers optionality of relationship; any entity type A which participated in a relationship of the form:

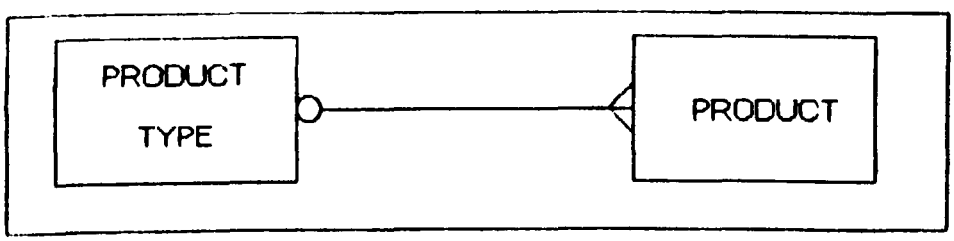


would be a principal entity type. Ellis [ELLIS, 85] showed that about 94% of relationships would meet this criteria!

Information Engineering itself had a previous concept, called 'subject' entity type. This is quite close to the major entity type concept, being the central point of a subject area (see C3.4) and defined as a major resource or active entity type [JMA, 86a]. This definition is too restrictive as some major entity types are not major resources or active; for example, one actual example of a major entity type has been Gas Type - it is the gas which is the resource in this company, Gas Type is just a classification of this. In another respect the definition is too loose as it encompasses such entity types as Order and Delivery, which are neither fundamental nor shared.

In general a major entity type is one which has no 'crows feet' on it, ie. is always the 'owner' in a relationship and affects more than one functional area. They are often characterised by having an inherent stability and an easily discernable, though often complex life-cycle. Building on Rosenquist, it should also meet one of his properties - ie. it should not have any mandatory relationship memberships. That said, it is allowable for a major entity type to be mandatorily and 'many' related to another major entity type.

When considering whether an entity type is major, care must be taken with external objects (see appendix X2) and classificatory entity types as these can give strange results due to their properties. External objects tend to be major entity types anyway due to their significance to the business area. Classificatory entity types may also be major if their classification properties are used in a number of functional areas. If not then they should be ignored for the purpose of determining major entity types. As an example, a classic situation is:



Product is almost always major, but would not be by the above description unless Product Type was itself major (which it occasionally is) or were ignored.

C3.4 SUBJECT AREAS

Subject areas are collections of entity types, often gathered round one or more major entity types, the subject entity types. In fact subject areas are aggregation abstractions of entity types.

The subject areas represent broad areas of an enterprise and are analagous to functions. Examples of subject areas are Ordering and Distribution.

Subject areas can be decomposed into smaller subject areas and can be aggregated into higher-level subject areas. Subject areas can also contain major entity types when all the minor entity types related to a major entity type are contained in a single subject area at a level (they must be distributed over more than one subject area at a lower level for the entity type to be considered major).

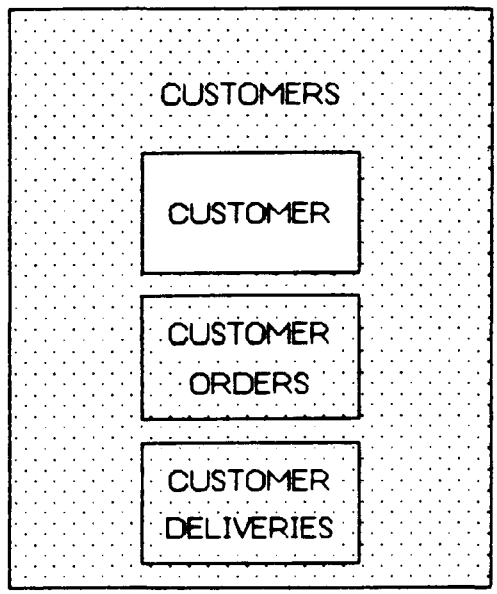
The aggregation is nominally just the logical horizon of the 'lowest-level' aggregated object. So subject areas nominally contain the logical horizon of entity types or subject areas. I say nominally because entity model clustering needs to allow

various oddities that occur in real life to be a pragmatic technique.

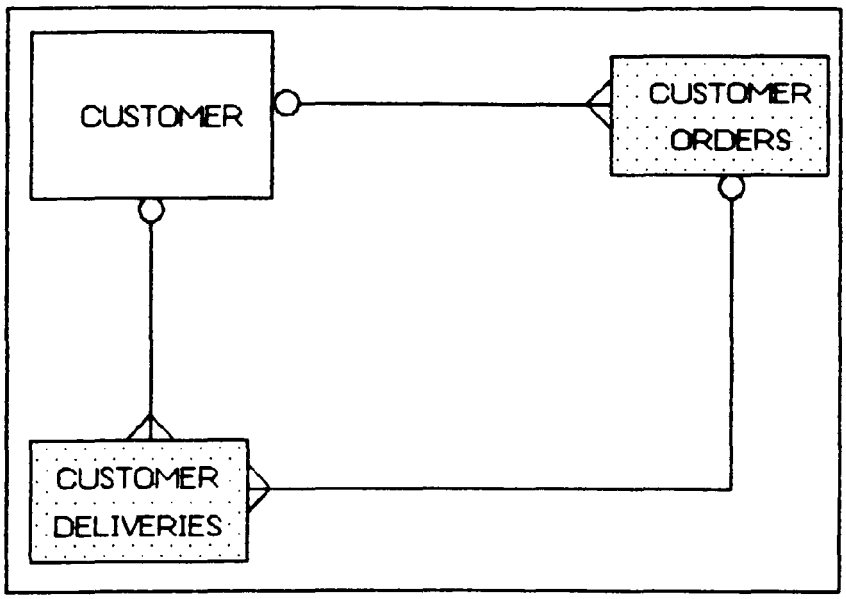
Subject areas were in Information Engineering before entity model clustering. However it was ill-defined as a concept. It was thought that subject areas did not decompose into other subject areas, but only into entity types. They were thought to be centred on a single subject entity type (see C3.3), which was not necessarily major and had little use outside of the initial identification of global data.

Entity model clustering developed a rigorous understanding of subject areas and of their constituency. Basically, subject areas can be thought of as decompositions of the relationships between major entity types. This bears a correlation to a method used to analyse information requirements, which is to find a high-level 'view' and to continuously decompose the relationships between the entity types found. In conventional models, the constituents of this decomposition can only be documented as a non-hierarchical diagram. Clustered entity models are mainly formed by clustering these non-hierarchical diagrams back into a hierarchy by the use of abstraction; the clustering process is guided by the major entity types previously found.

A high-level subject area consists of subject areas and major entity types, eg.



This subject area can be decomposed one level to separate out the subject entity types, as in:



where Customer is a major entity type and Customer Orders and Customer Deliveries are lower-level subject areas. As Customer 'belongs' to the higher-level subject area, it is not related to any other entity types in this environment.

Subject areas can continue to be decomposed into succeedingly more detailed subject areas until a subject area is decomposed into its constituent entity types. In the above example Customer Orders and Customer Deliveries are probably at the lowest-level and would decompose straight into minor entity types.

Some major entity types just affect a small part of a larger subject area; in which case they will be separated out when that part of the area is itself decomposed. As an example, Customer

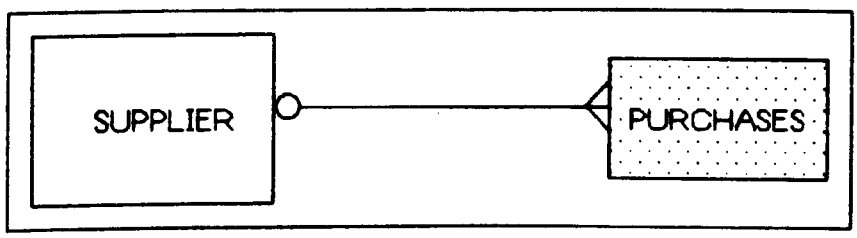
Type may be fundamentally important to Ordering, but no other part of an enterprise. If this were so, Customer Type would appear in the Customer Orders subject area and be separated out when this is decomposed.

The aggregation to form subject areas is considered in the next section, C4.

C3.5 RELATIONSHIPS IN A CLUSTERED ENTITY MODEL

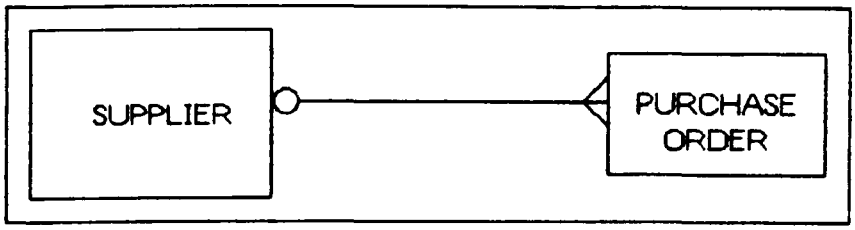
Relationships are the same as in a non-clustered entity relationship model. However, there are some restrictions, but some increased flexibility as well.

Relationships can be shown from a major entity type to a complete subject area, as in:

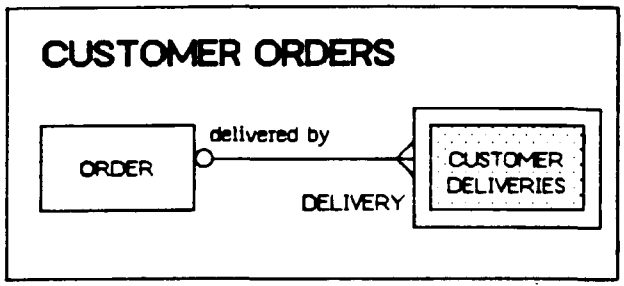


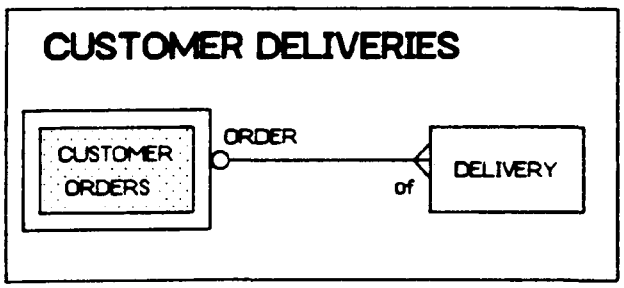
The cardinality and optionality are derived from an aggregation of all the relationships from the major entity type to the subject area.

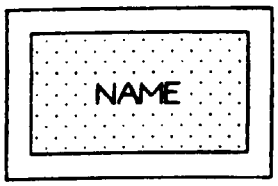
On the diagram corresponding to the related subject area, the relationship is shown to the appropriate non-major entity types, for example:



Where there are direct relationships between subject areas, ie. relationships with no major entity type intervening, the relationship is shown to the whole subject area, but labelled with the name of the related entity type. This is shown as:





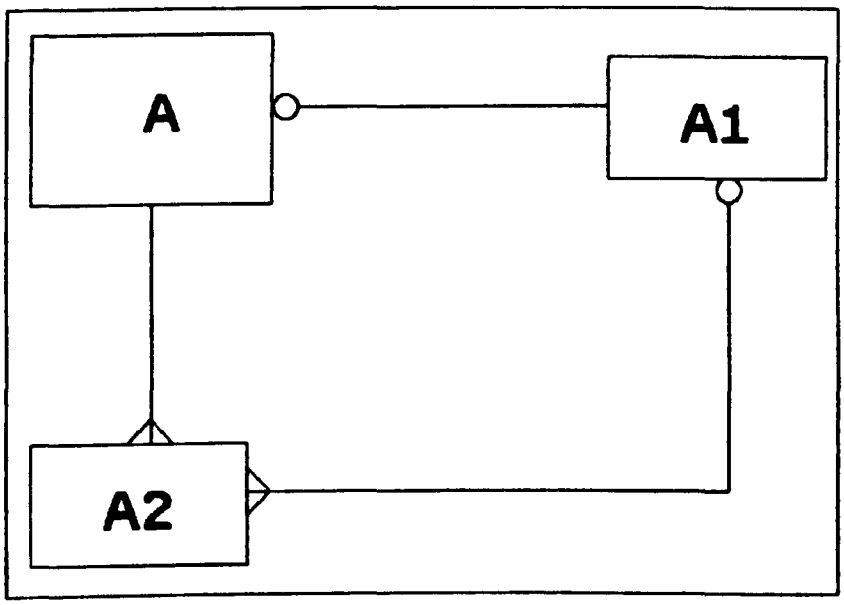
where 

is an inter-diagram connector to the named subject area (depicted on the same named diagram).

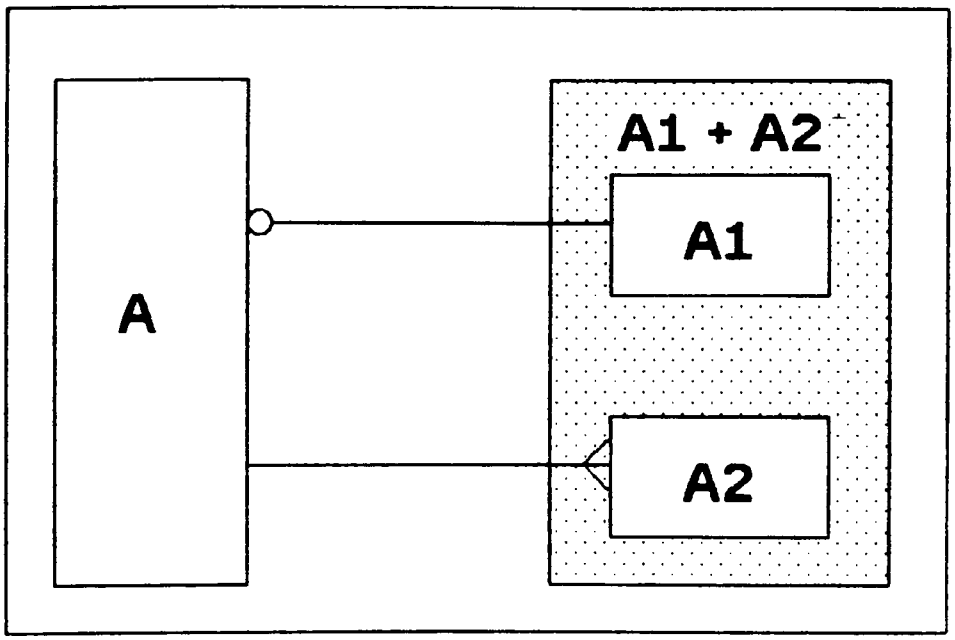
With relationships to subject areas, the optionality and cardinality is shown, but is aggregated from all the constituent relationships between the related objects.

The aggregation of relationships needs to be such that all the relationships still hold. To achieve this the least stringent of the relationships' properties should apply. So if there is any optional relationship, the aggregated relationship must be optional. Similarly if there is any many relationship then the aggregated relationship must be many. This is consistent with the definition of optionality and cardinality for relationships between two entity types; if ever an entity type can exist without the other it is optional and if ever there is more than one it is many.

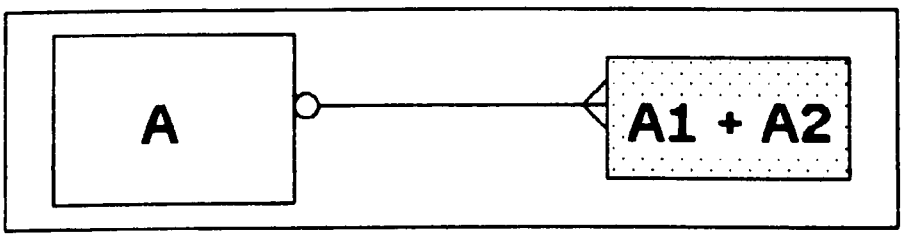
For example:



A1 and A are within the logical horizon of A2, so can form a first-level subject area.



If a single relationship is to be shown, it must be an aggregate. As the A/A1 relationship is optional, the aggregate must be optional; similarly as the A/A2 is many, the aggregate is many giving:



These aggregate relationships rarely have a meaningful name; they will only be named if all the aggregated relationships are the same.

This example showed aggregation for a major entity type and a subject area: the same discussion would apply for aggregations between two subject areas (two subject areas are related if there are entity types belonging to them which are related).

Relationships between major entity types are never aggregated.

C3.6 SUBJECT AREA DIAGRAMS

A diagram is drawn for every subject area at every level showing its consitituents and the major entity types which are related to it.

There will be one high-level subject area which covers the whole enterprise; as all the major entity types can only affect this subject area they will belong to it. The diagram of this subject area covers the whole enterprise in overview and is very useful for communication purposes as a scene-setter

The high-level subject area contains the major entity types and lower-level subject areas. Each lower-level subject area has a diagram of its own and may contain subject areas of its own. The diagrams at this lower-level are useful for focussing

attention on the aspects of each subject area.

Eventually there will be subject areas which decompose into minor entity types. This level is the bottom-level and is the most useful diagram because it shows the detail we are after. All the higher-level diagrams serve to set the context for this lowest-level diagram. Figure C3.1 shows a three-level hierarchy.

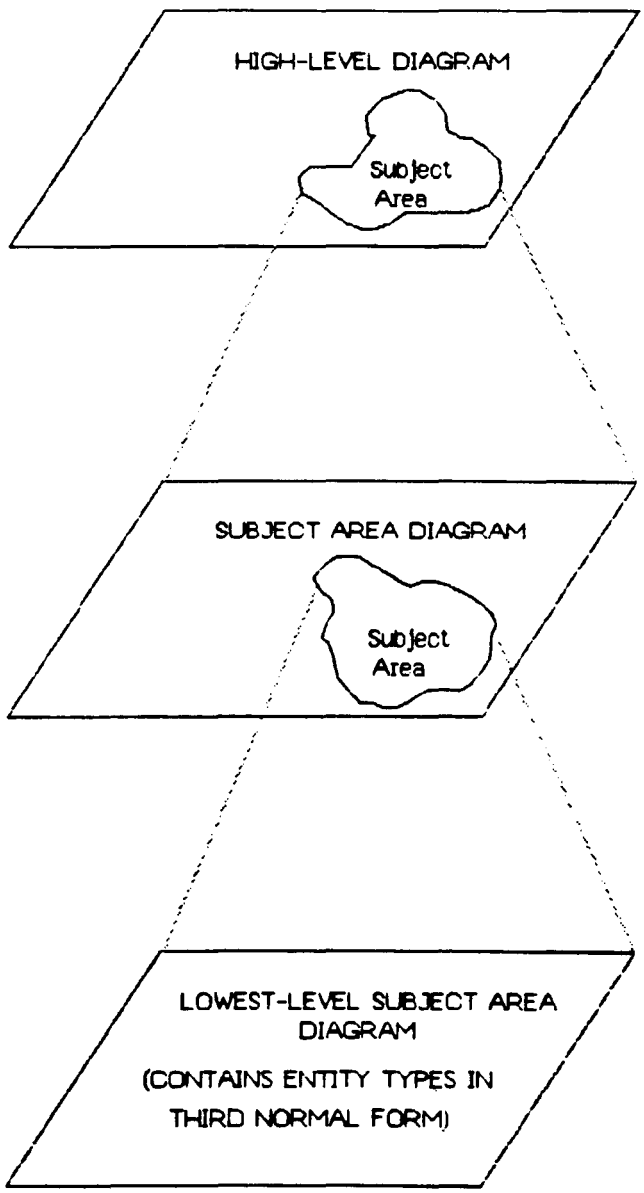


Figure C3.1 A Three-Level Subject Area Hierarchy

The lowest-level diagram is an entity relationship diagram of the entity types belonging to that subject area. The only differences are the inter-diagram connectors which show the relationships to other areas, and the delineation of major entity types.

Relationships between major entity types are not always shown on lower-level subject area diagrams, however they must all be shown on the highest-level diagram. Only those major entity type relationships which enhance the communication of an area will be shown. One reason for this is that it improves the maintainability of the model by-hand. The main reason is that some of these relationships have little or nothing to do with the area, so there is no point showing them where it is more likely to confuse.

A similar argument applies to subtypes of major entity types - they must all be depicted on the highest-level diagram, but only those which affect a given subject area should be shown on lower-level diagrams.

Examples of subject area diagrams are contained in C5.

On the diagrams it is useful to distinguish between major entity types, subject areas and minor entity types. They all have square boxes as they are all data (see appendix X1). The conventions found to be best are to shade subject areas and

major entity types (differently from each other) but not to shade minor entity types. If it can be guaranteed that all the boxes can be the same size (this is difficult to achieve, eg. if subtypes are shown, larger boxes are often needed) then another convention that works is to use larger sized boxes for major entity types than minor entity types and to show subject area boxes with thicker edges than entity type boxes.

One of the guiding aims in the development of entity model clustering was to ease maintenance, ie. the upkeep and change of diagrams. As there are many fewer components on any single diagram, it is much easier to draw the different, smaller diagrams than when all entity types appeared on a single diagram with a large number of relationships. However, the danger of proliferating and duplicating entity types had to be avoided. This was achieved by constraining each minor entity type and subject area to appear in only one place but be referenced as many times as necessary by means of inter-diagram connectors. The inter-diagram connectors are duplicated at source and destination to enable all connectors to a component to be easily found without complication.

C3.7 CARTOGRAPHICAL ANALOGY TO THE USE OF A CLUSTERED ENTITY MODEL

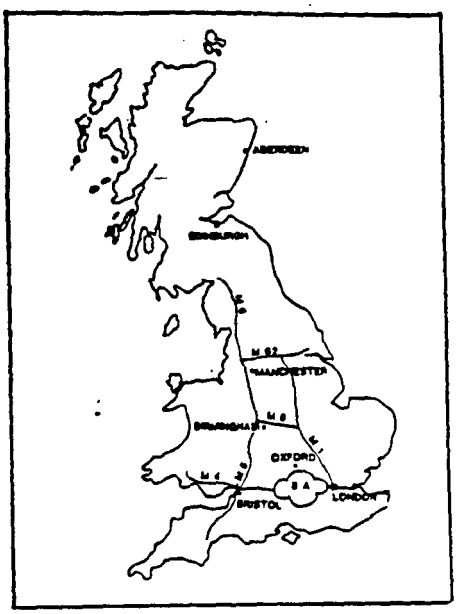
The approach we proposed here for the management of large entity relationship models is analogous to the cartographical method of locating addresses within a country (see figure C3.2). A

traveller unfamiliar with the geography of a country would first use an overview map to locate the desired area, making use of major cities as landmarks. The traveller would then resort to a more detailed map to locate a town in relation to the cities and to a street map to actually pinpoint a required address. Roads and rail networks correspond to relationships between cities, towns and so on.

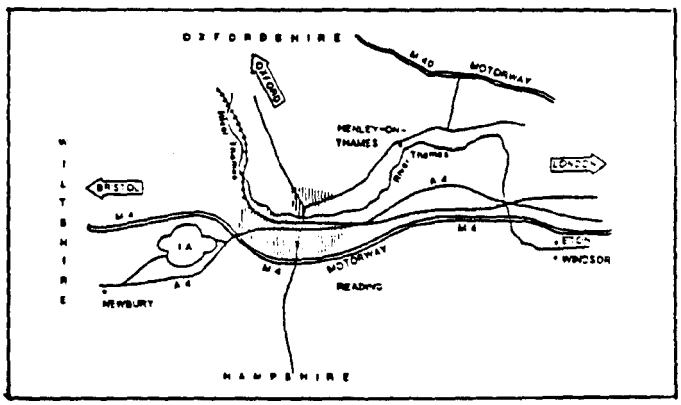
If a street map were viewed without previously consulting smaller scale maps then the beneficial context of the town would not be available; for instance, the best route to take and the geograpical nature of the area, eg. whether in north or south, in desert or rainforest, cannot easily be found from street maps. Also, without this context there is always the possibility of using the wrong map, eg. a map of Washington, England or Washington State, USA instead of Washington DC, USA.

The analogy to clustered entity models is that a map of a country is equivalent to a high-level diagram, a map of a county or state to a higher-level subject area, and town plans to lower-level subject areas. Town landmarks correspond to entity types, cities to major entity types, town streets to relationships between the landmarks, inter-town roads/rail to inter-diagram connections, and town/city connections to relationships between minor and major entity types.

High-level Diagram of Great Britain



Subject Area Diagram of Berkshire



Subject Area Diagram of Bradfield, Berkshire

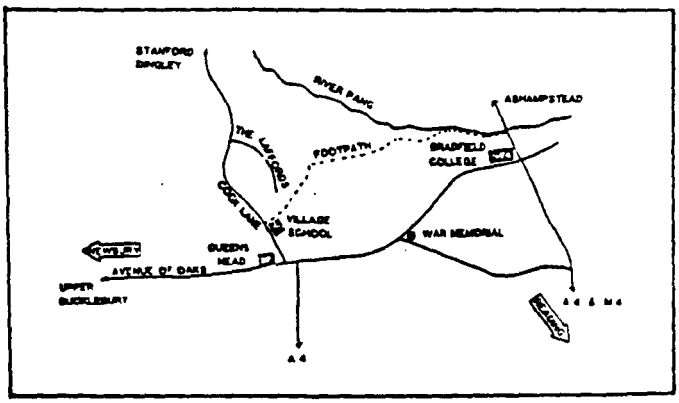


Figure C3.2 A Cartographical Analogy to a Clustered Entity Model

C3.8 ABSTRACTIONS ON ENTITY RELATIONSHIP MODELS IN ENTITY MODEL CLUSTERING

Entity model clustering does not impose any restrictions on the way that abstraction is dealt with by entity relationship modelling. In fact the clustering process is a controlled form of aggregation abstraction - all of the entity types within an area are aggregated to form that area. In entity model clustering, minor entity type subtypes are dealt with in exactly the same manner as for conventional entity relationship modelling, but a degree of flexibility is introduced in the depiction as subtypes of major entity types.

Due to their nature, major entity types tend to have a complex structure. This is often shown by the use of involuted relationships and subtypes, for example, an Organisation major entity type would have a hierarchy, represented by an invoulted relationship, and consist of many mutually exclusive parts, represented by subtypes. Involuted relationships should be shown throughout the model, however subtypes only need to be depicted on the highest-level diagram and in those areas which they apply to. For example, assume major entity type Product consists of Bought Product, Sold Product and Intermediate Product subtypes. Product would be shown wherever necessary, however the subtypes would only be shown on those areas where they are applicable, for instance, Bought Product would be shown on a purchasing area and Sold Product in a sales area. In these cases the inapplicable subtypes could be generalised to Other Product.

Thus the view of a major entity type can change depending on the area under consideration. The basic meaning of a major entity type remains static throughout the model, it can just be interpreted in different ways in different contexts.

Generalisation by relationships is unaffected by the use of entity model clustering. However generalisation hierarchies would be used in the determination of area boundaries as it is highly desirable to keep generalisation hierarchies intact. This is achieved because generalisation relationships have a structural nature with the components of the hierarchy invariably used in the same way. So a generalisation hierarchy would tend to appear in a single subject area. The same argument applies to aggregation.

C4 CLUSTERING AN ENTITY RELATIONSHIP MODEL

C4.1 METHODS OF FORMATION/DERIVATION

This subsection considers the most effective method of clustering an entity relationship model. This method is iterative and the results are adaptable to an organisation and its information requirements. The most likely application for the method is to structure an existing conventional model, or set of models. However it has also been used to guide the development of an entity relationship model. Only the former is considered here, the latter is discussed in C7. The method is empirical, but based on the tried and tested concepts of abstraction and decomposition (see appendix X1). There are two aspects to the formation of a clustered entity model, algorithmic (discussed in C4.2) and interpretive. The interpretive aspects arise from the peculiarities inherent in any model of human activity and derive either from the activity or from the modellers view of it, The method described here does not just apply to single models, it has been used on a related set of models with a single, clustered model being synthesised.

C4.1.1 Finding Major Entity Types

First the major entity types of the modelled organisation must be extracted. Based on Ellis's [ELLIS, 82] concept of logical horizon described in appendix X1, we can isolate major entity types. As discussed in C3.3 all of a major entity type's relationship memberships should be 1:MANY, optional 'outwards' (major entity type : related entity type). For example, in figure C5.1 all of the relationships to entity type Product are 1:MANY, optional with the 1 on an edge of the Product box. Furthermore, a major entity type should be of fundamental importance to more than one functional area of the organisation, ie. should appear in more than one subject area. For example, in figure C5.2 Customer has relationships to two subject areas. (These examples are expanded upon and their derivation explained in section C5.)

So major entity types are fundamental, shared data. They can be found by analysing a model to discover those entity types which have only 1:MANY outward, optional relationship memberships, or whose MANY:1 inward relationship memberships are only to major entity types. Care must be taken where MANY:MANY relationships appear on a diagram. For the purposes of discovering major entity types, MANY:MANY relationships can be treated as 1:MANY outward relationships as the majority of such relationships can be notionally broken down into 1:MANY relationships with

'intersection data'; it is a notional breakdown because it does not need to actually be done.

The set of major entity types formed from this process will be modified by removing any entity types which are only related to single subject areas. Other entity types can be added to this set if they are found to be of significant interest to a number of subject areas.

The two aspects of entity model clustering can be seen here; the relationships are made use of in an algorithmic fashion and the results are subsequently interpreted to deal with inbuilt anomalies in the base entity relationship models.

C4.1.2 Forming Subject Areas

As described in C3.4, a subject area consists of levels of logical horizons. So to form a subject area the logical horizons need to be found.

The first-level subject area is formed by finding the horizons of the minor entity types and then successively abstracting the logical horizons until there are only logical horizons and major entity types. These abstracted logical horizons are the 'first-cut' subject areas. More often than not, this process actually results in more than one subject area relating to the

same group of major entity types. These similar subject areas are then abstracted to form a higher-level area by using their logical horizons.

It may be necessary to continue this process to higher levels of abstraction for an organisation with very complex and/or diverse information requirements. There is no restriction placed on the number of levels which can be used. It has been found that the size and complexity of information requirements bears a close relationship to the diversity of an environment. This is because different entity types and their relationships are needed to cope with the diversity of information modelled. So for simple environments only two levels of clustered entity model may be appropriate, while for very diverse environments four or more levels may be appropriate. The number of levels should not be pre-determined, but should be allowed to grow with the model during the analysis process. The description above is based on the results of a study at Whitbread & Co Plc, where it was felt that three levels were a manageable number that did not cause overcrowding of diagrams.

C4.1.3 Inter-Subject Area Relationships

As considered in C3.5, relationships can exist between subject areas, ie. across area boundaries. These relationships arise when the clustering process results in more than one cluster

between major entity types. For example, in figure C5.1 Order appears in two logical horizons and hence, 'belongs' to two subject areas; to enable the easy maintenance of the model (see C3.6) Order can only appear in one subject area, so some of its relationships will have to cross the boundary to the other subject area. Boundary relationships occur when clusters are formed between groups of major entity types where some of these major entity types are duplicated across the groups; for example, Customer and Product appear in both the logical horizons Order appears in.

If a boundary relationship is found, a decision will need to be taken as to where to draw the boundary between the clusters, ie. which minor entity types/subject areas will appear in which subject areas. Clusters broadly relate to the use made of their entity types because a large number of the relationships reflect this use. Also, entity types which are related due to 'structure', e.g 'Order consists of many Order Items', tend to be strongly bound together and made use of in the same way in most processes. This is not so with entity types related due to other causes, eg. appearance in a common process as in 'Order results in Delivery'. As it is sensible to keep structurally related entity types together wherever possible, the boundaries between areas should be drawn to keep structural relationships in the same area. If there are no structural relationships, entity types should be put in the area they are closest to

functionally. For example, a Delivery entity type should be put in a distribution related area as opposed to a production related area. If the result of the decision is not obvious, the boundary may hide a new major entity type.

As for major entity types, some manipulation of the results may be necessary to accurately reflect the information usage and requirements of an organisation. This can result in more boundary relationships. However in these cases the above guidelines should still be followed.

The use which the clusters reflect is not that of detailed processes, but that of broad functional areas of the organisation. These functional areas are not necessarily related to the organisation's structure, quite often they cut completely across existing structures, but they do relate to the main purpose and activities of the organisation. For example, an organisation with a Purchasing function may not have a specific purchasing department, but may have purchasing distributed across all of the organisation's departments. If there is a Purchasing function, there will probably be a Purchasing subject area which clusters the data used by the Purchasing function. On the other hand, 'Buy Equipment' and 'Decide on Supplier Suitability' subject areas are unlikely to be found; both these processes are part of the Purchasing function and would use the data in, or related to the Purchasing subject area.

A knowledge of the functional requirements of an organisation is useful to decide on boundary relationships and to name the clusters formed. The subject areas are named to reflect the functional area they relate to (eg. Personnel) or to reflect groupings of data (eg. Ordering and Delivery). Occasionally clusters are concerned mainly with a particular major entity type, in which case they are named appropriately.

C4.2 ALGORITHM FOR CLUSTERING AN ENTITY RELATIONSHIP MODEL

An algorithm for clustering an entity relationship model was constructed. It is documented by means of an action dependency diagram (see Chapter B for a description of this technique), backed up by an action diagram.

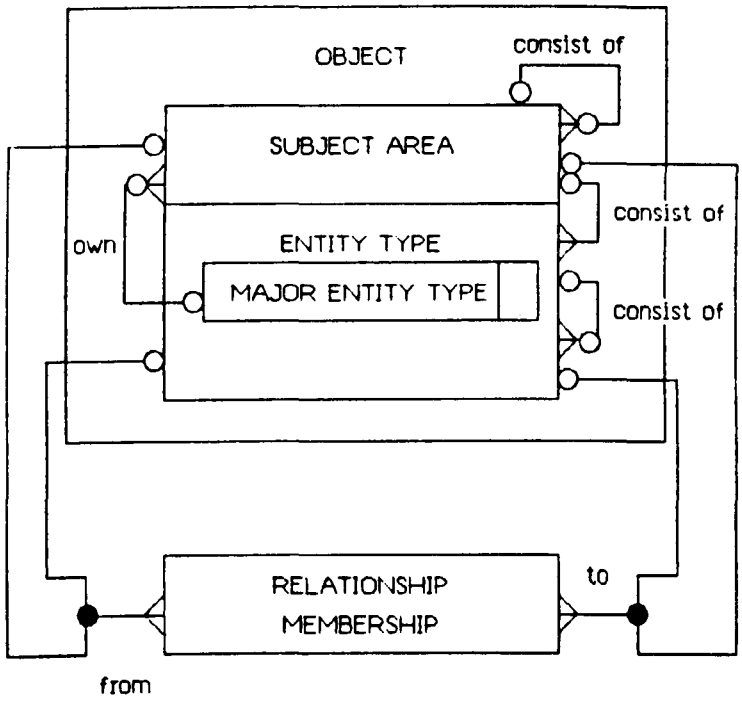
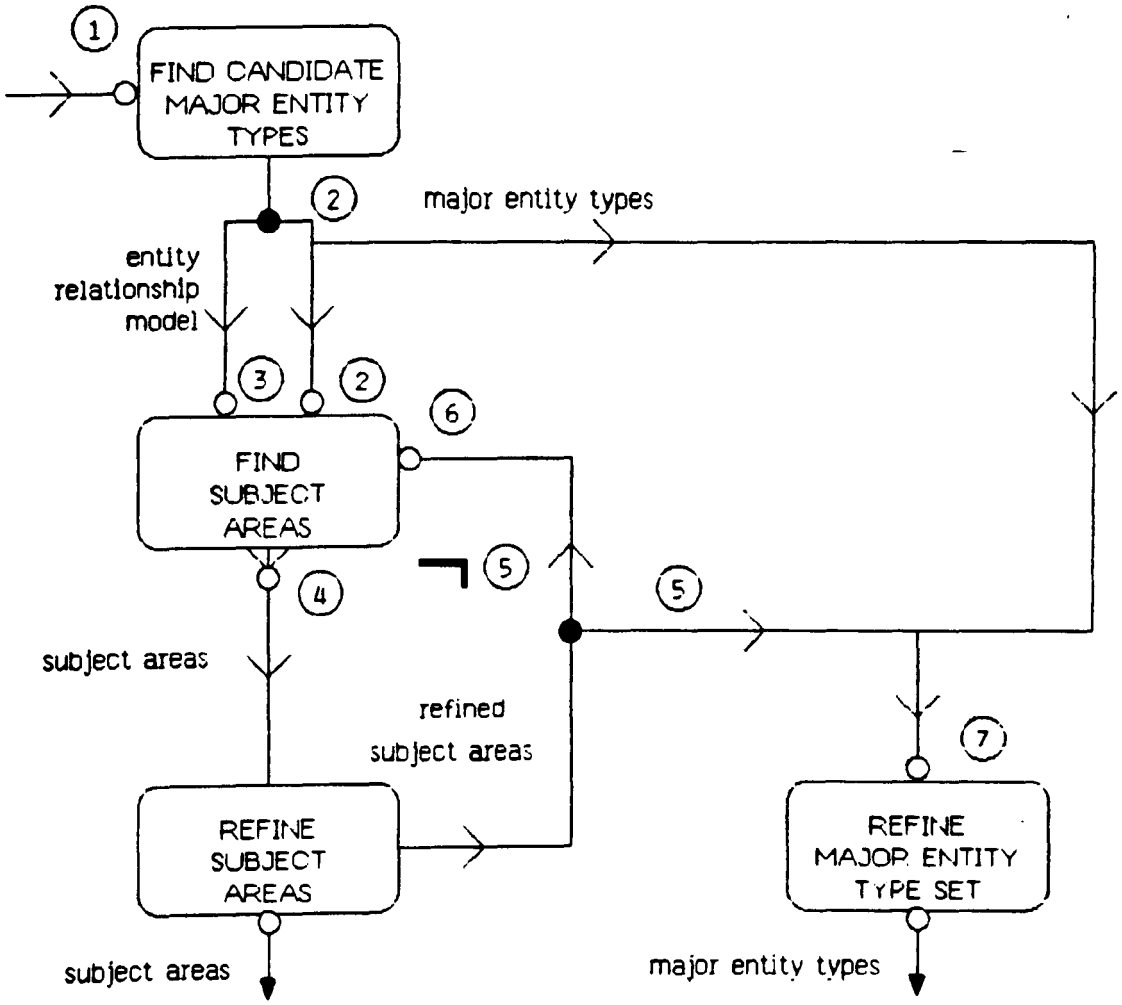


Figure C4.1 Entity Relationship Diagram of Clustering Objects



(1) Entity relationship model with > approx. 20 entity types

(2) Major entrity types found

(3) Major entity types not found

(4) All subject areas formed

(5) <= 7 subject areas found

(6) Not first time

(7) Subject areas found & (2)

Figure C4.2 High-level Action Dependency Diagram of Clustering

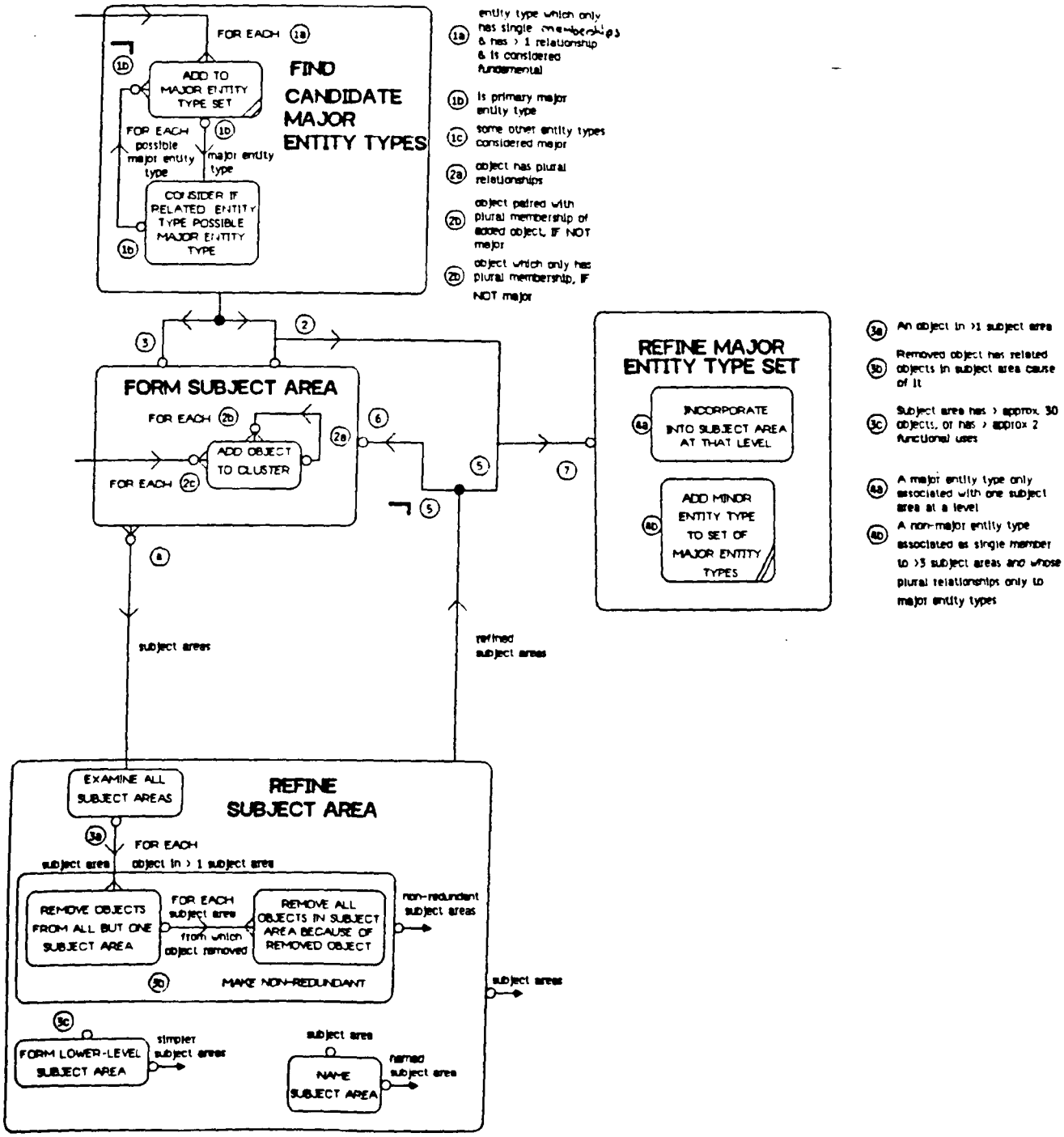


Figure C4.3 Detailed Action Dependency Diagram of Clustering

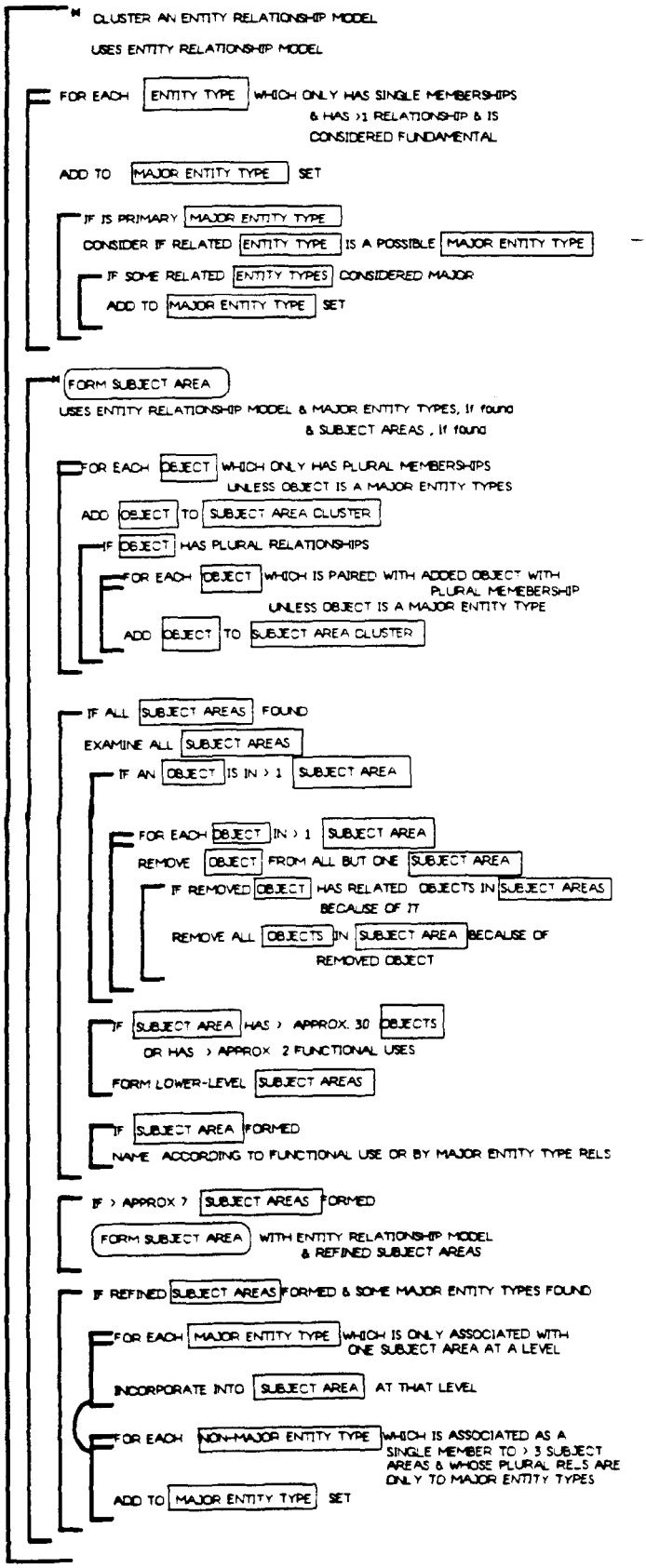


Figure C4.4 Action Diagram of Clustering

The algorithm enables a first-cut clustered model to be formed, however care must be taken to interpret the results to achieve a communicable model. The importance of the communication potential of the model should always be paramount.

C4.3 PRACTICAL GUIDELINES

1. A subject area generally has the characteristics of being the consolidation of the logical horizons of the 'lowest-level' entity types in the subject area (ie. those which only participate in plural memberships).

2. Major entity type should only be single members. If they have plural memberships then these should be to other major entity types.

3. Ideally a subject entity type is an entity type which is related as single member to more than one subject area as it should affect a number of parts of an enterprise (ie. is 'shared') and should be a determining factor in those parts.

   Some major entity types are not as obvious as others. Again the choice of major entity types is often context dependent.

4. The most important criteria for a major entity type is that it is something fundamental to the enterprise (ie. is considered to be fairly important to the running of the enterprise). This should override all other criteria (eg. may have plural memberships). However this is a subjective criteria.

5. If a major entity type is only related to a single subject area at some level then incorporate the major entity type into that subject area.

6. The exact contents of a subject area is dependent on the context in which the clustering is carried out. The main problem is deciding which subject area an entity type belongs to where it may appear in more than one. There are no hard and fast rules here.

   Where a non-major entity type is related to more than one subject area then assign it to the subject area to which it has the stronger relationship, i.e. which contains other entity types that have the stronger relationships to it. Basically a strong relationship is one which is due to structural reasons as opposed to functional reasons.

7. Ideally subject areas should contain about 7 other subject areas, except at the lowest-level where they should contain

20-30 entity types, for the best comprehension of the complete model. However some lowest-level subject areas will contain only one or two entity types - two is generally indicative of an area which has been insufficiently analysed (though maybe for good reasons such as the area is outside the scope of the project). Some lowest-level subject areas may contain more than 30 entity types. These should be examined as there may be more than one subject area, but it might be a genuine case where there is a large number of different types of things classified as being about the same subject.

There is a temptation to produce small subject areas when forming clustered entity models. This can be problematic at the lowest-level, where the non-major entity types are found, as it becomes difficult to present and even more difficult to comprehend the complete model if the subject areas at this level have few objects.

8. When forming a clustered entity model, assign each entity type to a single subject area (it may be moved to a different subject area when further analysis has been done). Doing this improves the maintenance of the model as the impact of changing that entity type can be easily found by just examining its subject area.

9. The main purpose of clustering an entity relationship model during Business Area Analysis is for presentation. If the business require an entity type to be major, or think that it should be in a different/related subject area then don't argue too much. After all it's their model and them being presented to.

10. The initial drawing of a clustered entity model typically takes twice as long to produce as a diagram of a non-clustered entity relationship model. However, this time will be saved later on in a project as the maintenance of a clustered entity model is much quicker. For example, if an entity type is removed from a non-clustered entity relationship diagram this may necessitate the re-drawing of that complete diagram; in a clustered entity model only the affected subject area diagram would need to be redrawn and any inter-diagram connections to that entity type removed.

C5 AN EXAMPLE OF ENTITY MODEL CLUSTERING

This section considers an example of forming a clustered entity model from a conventional entity relationship model. The conventional entity relationship diagram which is the basis for this clustering is shown on figure C5.1. This diagram covers ordering, distribution, stock handling and staff handling in a simplified form to enable the example to be more easily understood. Note that the diagram bears no relation to any actual enterprise. For the purpose of this example some logical horizons have been delineated on the diagram, these would not normally appear. The results of the clustering process are shown in figure C5.2. The algorithmic and interpretative aspects have not been explicitly separated because they are too inter-dependent in this example.

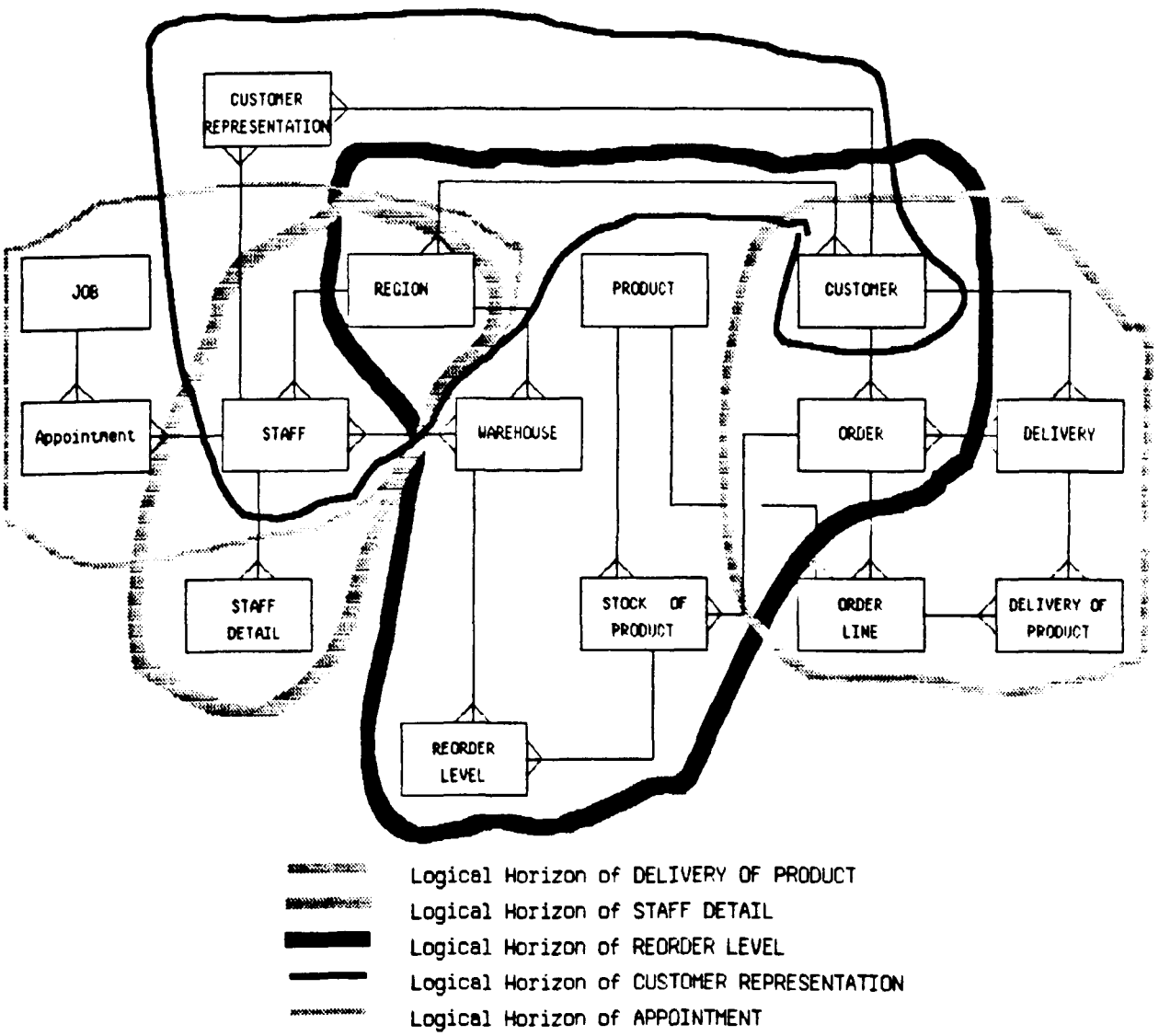
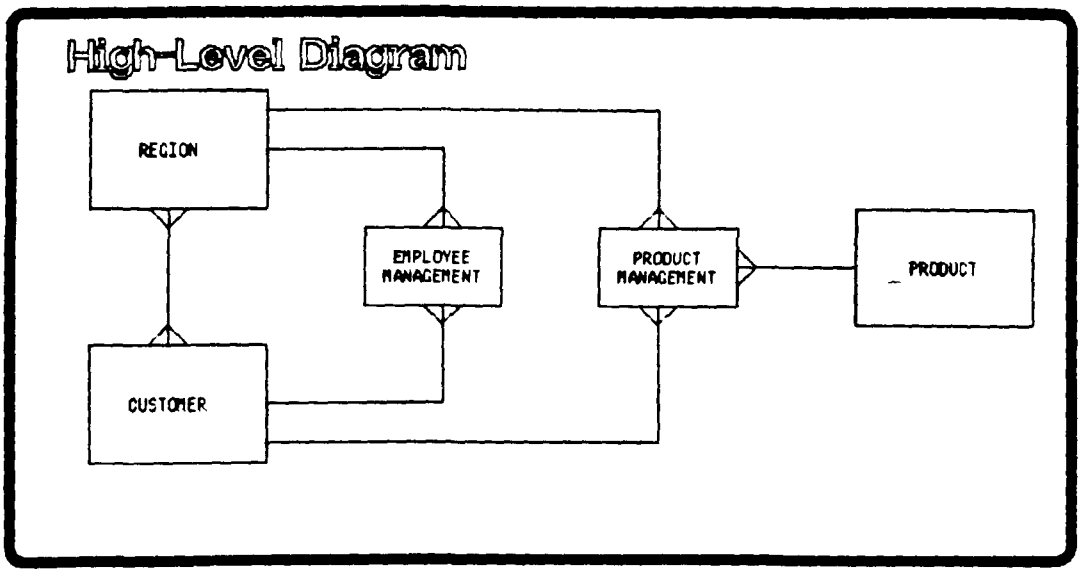
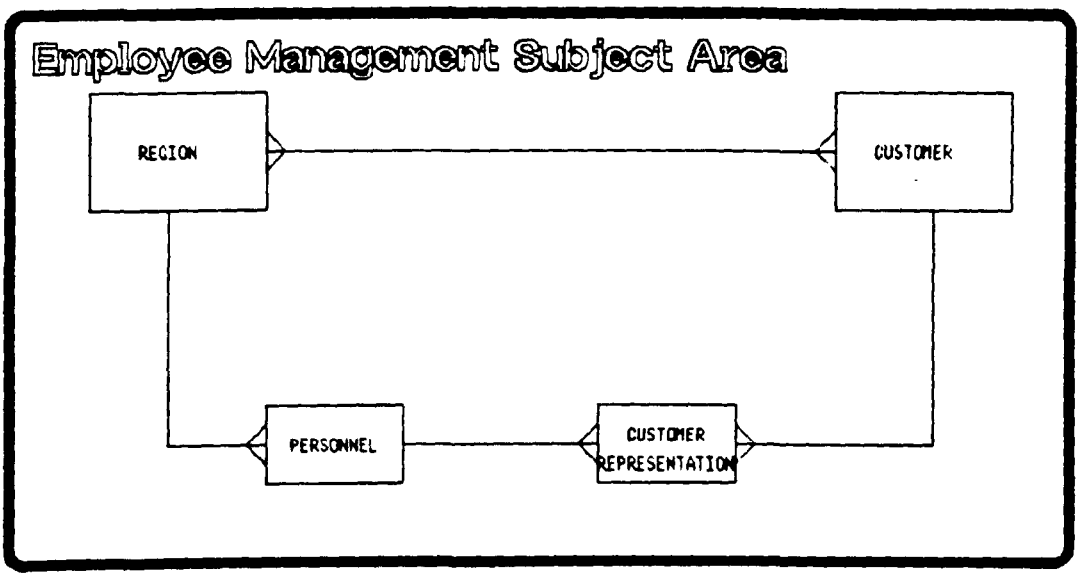
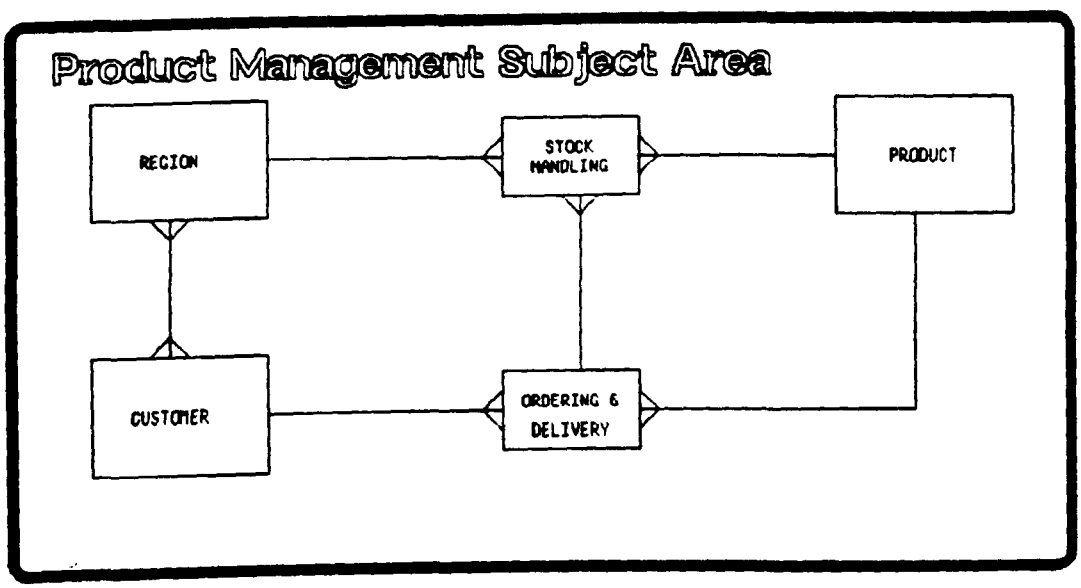


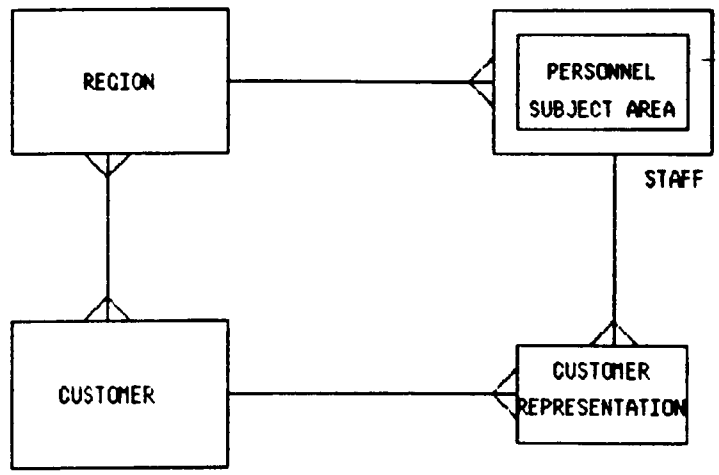
Figure C5.1 Example Entity Relationship Diagram Showing Logical Horizons


High-Level Diagram
REGION
EMPLOYEE MANAGEMENT
PRODUCT MANAGEMENT
PRODUCT
CUSTOMER

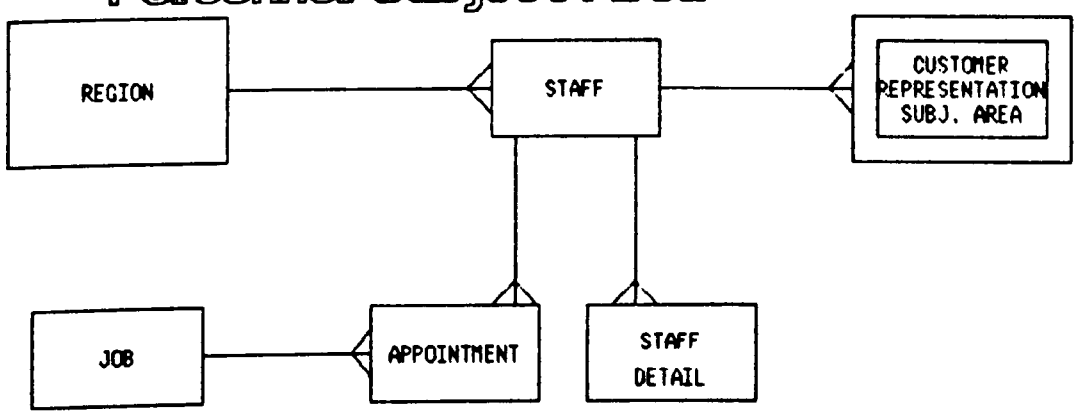

Employee Management Subject Area
REGION
CUSTOMER
PERSONNEL
CUSTOMER REPRESENTATION


Product Management Subject Area
REGION
STOCK HANDLING
PRODUCT
CUSTOMER
ORDERING & DELIVERY

Customer Representation Subject Area



Personnel Subject Area



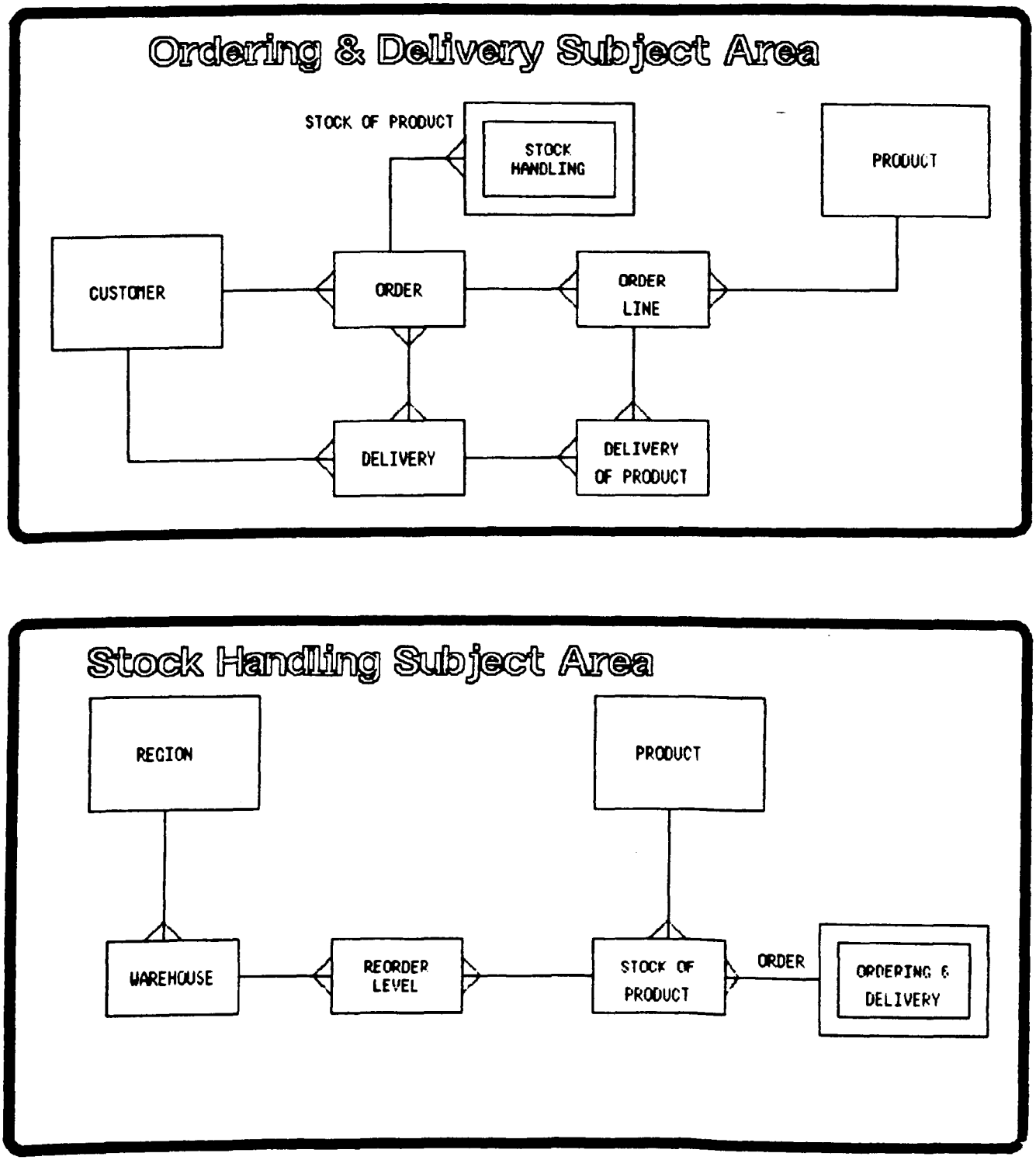


Figure C5.2 Example Clustered Entity Model of Figure C5.1

C5.1 FIND MAJOR ENTITY TYPES

The main criterion for an entity type to be considered major is that it only has 1:MANY outward relationships, ie. is at the 'top' of a logical horizon. From the logical horizons marked on figure C5.1 we can see that the entity types Customer, Job, Product and Region are candidates for major entityship (the MANY:MANY between Customer & Region being notionally decomposed into 2 1:MANY outward relationships). On consideration of these entity types, they appear to be a reasonable set of fundamental entity types. If we consider the criteria that a major entity type can have 1:MANY inward relationships from other major entity types, most other entity types would appear to be candidates; this is a result of the simplified example. Intuitively, the only other likely fundamental entity types are Staff and Warehouse.

Some people would expect Order and Delivery to be fundamental. However these entity types are based in the functionality of an organisation. This suggests that they are not fundamental because an organisation's activities themselves tend to be based around major entity types, for instance, Customer Handling and Product Production, both of which result in Orders and Deliveries. Customers and Products can exist without Orders and Deliveries, for example, new Customers and Products, but the reverse does not hold. This is not to imply that Orders and Deliveries are not important, in many ways they are more

important than Customers and Products, they are just not fundamental.

Therefore the intial set of major entity types is Customer, Job, Product and Region, with the possible inclusion of Staff and Warehouse.

C5.2 FIND SUBJECT AREAS

To find subject areas we will use the logical horizons delineated on figure C5.1 as the basis of abstraction and we will use the major entity types found above.

There are a number of start points for abstraction, all of them are the 'lowest' point of a logical horizon.

Take Delivery of Product:

At the top level there is Product and Customer; Order, Order Line and Delivery are in-between. This is one subject area which, guided by the contained entity types, we will call Ordering and Delivery.

Take Reorder Level:

Customer, Product and Region are at the top level; Warehouse, Stock of Product and Order are in-between. We will name this area Stock Handling. As can be seen, there is a boundary dispute between Ordering and Delivery and this area. The relationship between Order and Stock of

Product is 'fulfils', ie. is functional. The relationship between Warehouse and Reorder Level is 'has a', suggesting a structural relationship. The relationship between Stock of Product and Reorder Level is also 'has a' suggesting aggregation, which is structural as well. Therefore there should be a boundary drawn across the functional relationship between Order and Stock of Product.

Take Appointment:

At the top level there are Job and Region, with Staff in-between. The result of taking Staff Detail is a very similar set of entity types. This suggests that Appointment and Staff Detail should be in the same subject area; this we will call Personnel. The logical horizon based around Customer Representation has commonalities to this group in Staff and Region; this might suggest inclusion in Personnel, but because Customer Representation is related to Customer (a different major entity type) it probably needs to be a subject area on its own. The correctness of this decision can be seen if the type of processes that would apply to the different subject areas are considered. For example, Personnel would have employement-related processes such as Hiring, which would not directly affect Customer Representation, whereas Territory Selling processes applying to Customer Representation have little relevance to Personnel.

The MANY:MANY relationships between Staff & Warehouse, and Order

& Delivery need to be thought about. The MANY:MANY between Customer and Region is a relationship between major entity types, so does not affect the constituency of the subject areas. Order and Delivery only occur in the same subject area, therefore this relationship does not cause any difficulties. On the other hand Staff and Warehouse appear in different subject areas, so the relationship between them must be considered as a 'boundary dispute'. The question is whether or not the subject area containing Warehouse is the same as the subject area containing Staff. The resolution of the MANY:MANY relationship is the staff working at a given warehouse for a particular period of time. This suggests that Staff and Warehouse are not concerned with the same groupings, so belong in different subject areas. This can be seen to be a reasonable split if the broad functions represented by the respective subject areas are considered - Personnel and Stock Handling should be separate groupings. This latter, intuitive process has value because, if the areas had a lot of common functionality, the resolution of the MANY:MANY would probably have reflected this.

Thus we have four subject areas Personnel, Stock Handling, Ordering and Delivery, and Customer Representation.

C5.3 MAJOR ENTITY TYPE ITERATION

We must now examine the major entity types in the light of the subject areas discovered. An entity type can only be major if

it is shared between subject areas. Job only affects a single subject area so is not major. Product, Region and Customer all affect at least two of the subject areas, so are major. Of the two 'possibles', Warehouse only affects a single subject area, so cannot be major whereas Staff affects two subject areas, so can be major. Whether to include Staff as major or to have a relationship between the Personnel and Customer Representation subject areas is an interpretive decision. The latter is best for this simplified example; in a more complex example, where three or more subject areas are concerned, the former would probably be most useful.

C5.4 SUBJECT AREA ITERATION

In reality this example would be much too simple to justify another level of subject area, however some higher-level subject areas will be defined from the subject areas found to illustrate the entity model clustering concepts. The process is extremely similar to finding the first-level, so only broad reasoning will be given.

There are only four subject areas Personnel, Stock Handling, Ordering and Delivery and Customer Representation to start with. They are all inter-related. This intermediate situation is shown on figure C5.3. A logical horizon can be found which includes Stock Handling and Ordering and Delivery but excluding the other subject areas, similarly for Personnel and Customer

Representation. Thus there are two candidates for higher level subject areas. The boundaries between these areas can be resolved by using exactly the same reasoning as for the formation into the lower level areas. Furthermore, if the functional basis for the areas is considered, the groupings appear reasonable. So we have found two higher-level subject areas which will be named Staff Management and Product Management after their constituents.

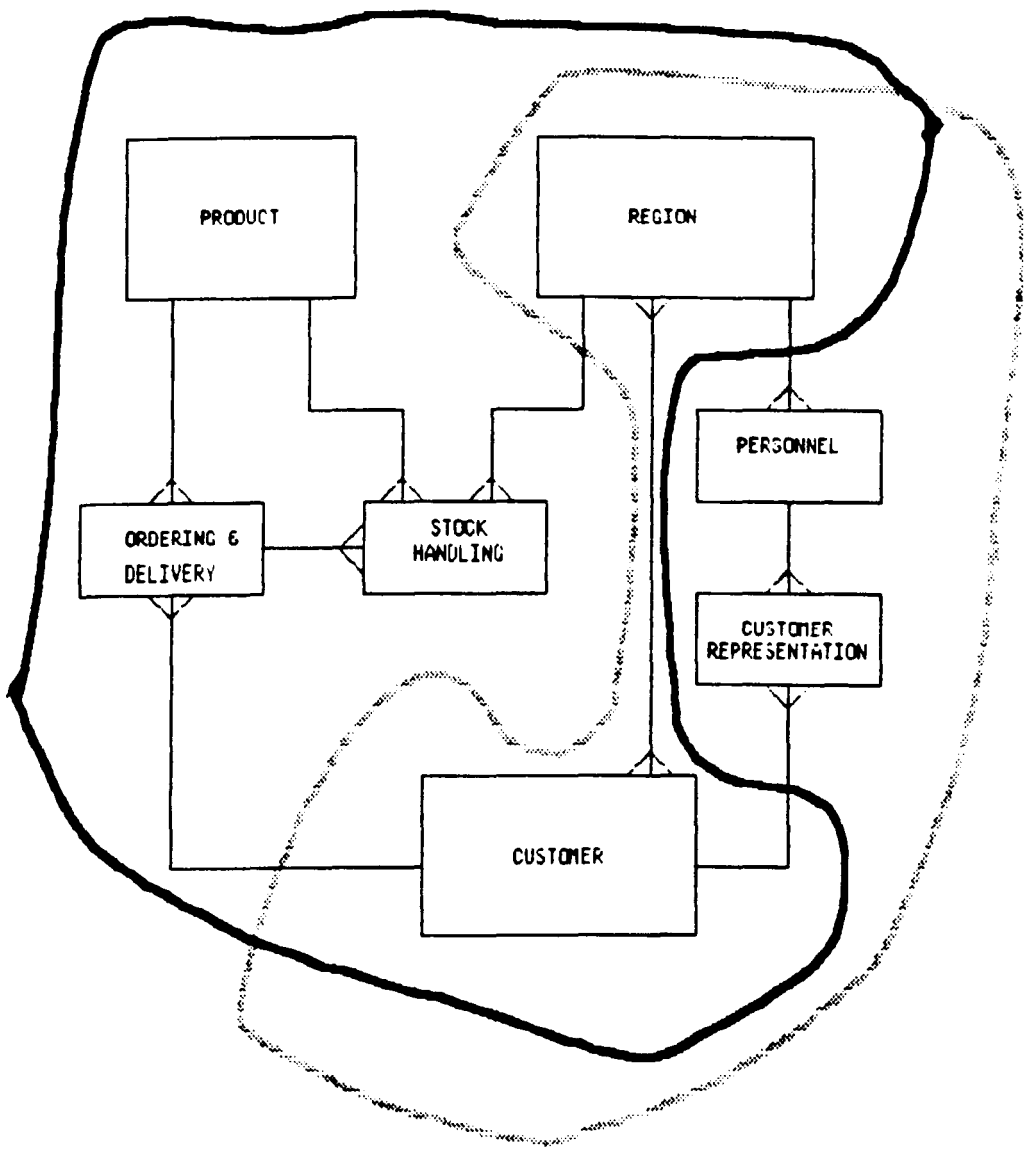


Figure C5.3 Intermediate Stage of Clustering Process

C6 DETAILS OF USE

Entity model clustering was developed in a practical context and has been used since in many other situations. Some of the companies where it has been used are:

* Whitbread & Co Plc
* Prudential Assurance Co Ltd
* International Paint
* Sedgwick Insurance Brokers Ltd
* Calor Gas
* Northern Gas
* James Martin Associates

at the time of writing. All of the uses have been successful, with the main benefits of entity model clustering and clustered entity models having been realised.

Unfortunately, due to confidentiality reasons, few actual results can be given, just a discussion of the implementations.

C6.1 WHITBREAD & CO PLC

Entity model clustering was developed at Whitbread & Co Plc from April-May 1983 to solve the problems discussed in C2. The cause of the development was that they had developed six strategic

'functional' views of Whitbread, which were cross-dimensional as Whitbread was organised geographically at the time. Whitbread required all the views to be consolidated and then made into a form where the results could be presented to the board of directors. As it was thought that the directors wouldn't appreciate a 150-200 entity type entity relationship diagram something had to be done. Thus was born clustered entity modelling. When the individual views were synthesized a clustered entity model was formed intuitively. It consisted of a set of major entity types (basically Customer, Site, Location, Organisation Unit, Geographical Area, Supplier, and Product) and two levels of subject area - an overview subject area and detail subject areas. Miller describes the results in [MILL, 85].

This model proved highly successful. I took the results away and developed the clustering algorithm based on what we had done intuitively.

In April 1984 Whitbread asked me to go back to add four more functional views to their synthesized corporate model giving a model of about 500 entity types, with about 30 of these being major. There were now three levels of subject area due to the size. In one 'leg' there were four levels of subject area to cover a particular part of Whitbread which had been covered in greater depth than the rest.

Also at this time Whitbread reorganised itself along functional

lines approximating the views taken the previous year. I suggested that it would be possible to produce views of the corporate model for each division and also for the original studies, using the lowest-level subject areas as building blocks. This was taken up and the views produced.

The divisional views were used as a basis for validating the underlying model as they formed a recognisable entry point for the users.

The Whitbread Corporate Data Architecture (what they call the result) is now well used within the company [MILL, 85]. I have since been back to Whitbread to advise them on the best use of the model and how best to maintain it. The model is used by their Data Adminstration function and is the source of data for all developments. An extract of the architecture is taken and developed; when the developments concludes the results are fed back to improve the model.

C6.2 PRUDENTIAL ASSURANCE CO LTD.

In 1984 I was involved in an analysis project at the Prudential Assurance Comapnies (PAC for short) General Branch operations. These involve such things as Commercial Insurance (Fire, Theft, Liability, etc), and Non-commercial Insurance (Motor, Non-Motor, Fire, Theft, etc).

When I was interviewed by PAC for inclusion in the analysis team I described the clustering technique and its results at Whitbread. They requested me to use it on their analysis. This analysis was split into three teams. Halfway through the analysis the results were synthesised by the project leader and myself and then clustered using the algorithm. The results of this were then used to guide subsequent analysis efforts, for example, the clustering highlighted that little claims information had been gathered at that time; claims is a vital part of the General Branch's operations and a team was directed to investigate it.

The clustered entity model was then updated at regular intervals to reflect the latest knowledge found. Little change was needed to the initial structuring, some entity types changed from minor to major and vice-versa, and more subject areas were added. The results proved useful and communicable.

The development was tasked to decide if an application package would suffice or if a bespoke construction was required. A study team was set up while the analysis was underway to make broad recommendations in this respect. The study team had little time to do this. Clustering proved invaluable here because it provided rigorous groupings of data to make decisions about, while allowing detail to be used where needed.

I have not been back to PAC to see the further use of the

results of the analysis.

C6.3 INTERNATIONAL PAINT

The development at International Paint was described in B6. Entity model clustering was used to structure an initial entity relationship model and highlighted broad areas of potential development. This was one of the inputs into defining the scope of the study as it enabled the functional use of data to be easily seen.

About halfway through the analysis the development was phased to try to achieve a quick result for political reasons. The data content of each phase was loosely based on a clustering of the entity relationship model they had at the time. Each cluster on the model formed a phase. A separate phase was designed to cover work-in-progress monitoring; it used data from all of the other phases, so couldn't be implemented until these phases had been completed. (Work-in-progress monitoring was one of the main purposes of the development.)

Clustering was also used to achieve a communicable model for the development as they have spent a large amount of time presenting their results to various parties, including the board of directors.

C6.4 SEDGWICK INSURANCE BROKERS LTD

Sedgwick's is one of Britains largest insurance brokers. They were undertaking an information strategy review with James Martin Associates help; I was called in specifically to use entity model clustering as part of this effort.

I had two tasks here, one was to cluster their global entity relationship model to achieve the benefits that would ensue. The second was to cluster an existing analysis model of a project which was under construction so the analysis model and the global model could be made consistent and any problems with the analysis model highlighted. Entity model clustering was used here because it enables inconsistencies to be highlighted both in overall structure (eg. a difference in what is considered major) and in detail (eg. if two subject areas dealing with the same broad functional use have different content there is an inconsistency). With the detail it would be very difficult to compare conventional entity relationship models because of problems in isolating the objects being compared. Overall structure is even more difficult to compare in conventional models as it is not identified.

The results highlighted consistencies and inconsistencies and the effort proved worthwhile.

C6.5 CALOR GAS & NORTHERN GAS

At Calor Gas and Northern Gas entity model clustering was used without my assistance so I do not have details of the results.

Calor Gas was a clustering of a strategic model. The results were good and Business Area Analysis projects are currently underway using the results.

Northern Gas carried out a Business Area Analysis of their 'Supplies' (every Supplier that they are interested in). After the analysis was complete, entity model clustering was applied to their entity relationship model because it was felt that the model was too large to understand and, hence, to design from. The result proved to be useful in enabling understanding to be gained of the model and was a good basis for designing a hierarchical database as the subject areas closely approximate the hierarchies required.

C6.6 JAMES MARTIN ASSOCIATES

In 1985 I was requested to quickly analyse James Martin Associates products operations. As part of this I formed an overview entity relationship model, clustered it and documented the result on Excelerator [INTE, 86]. The analysis was not completed so I have no documentable results.

C7 BENEFITS OF ENTITY MODEL CLUSTERING

Entity model clustering solves the dilemma of choosing between a large unstructured diagram that lacks cohesion and a superficial overview diagram that has insufficient detail. It enhances conventional entity relationship modelling techniques, enabling them to be applied to large scale and/or diverse problems without the difficulties described in C2. There are many benefits deriving from this ability.

C7.1 HIGHLIGHTS MAJOR ENTITY TYPES

The fundamental entity types of an organisation are highlighted by entity model clustering, the major entity types. This knowledge is essential for validation and communication. It is also essential for design as the fundamental entity types tend to require a lot of consideration when designing databases, eg. for access paths.

Invariably major entity types are very complex with involuted relationships, a number of subtypes and complex key structures. The highlighting of major entity types during the analysis process gives the opportunity to identify their structure and for their keys to be fully analysed. As an example of the benefit of identifying major entity types, it may be found that a full study is needed into a Product Code, but it is unlikely

that the key of an Order Item would require the same level of effort.

C7.2 STABILITY AND CORRECTNESS OF MODELS

The primary objective when developing entity model clustering was ease of use. This was achieved by allowing a major entity type to appear on any diagram where it is referenced by a relationship. This eliminates the cause of the majority of connectors and reduces the number of models that need to be viewed for any particular purpose. The structure enables an entity type to be identified quickly, in a top-down manner, without any prior knowledge of its name/synonym. Our second objective in the development of entity model clustering was ease of maintenance. This has been achieved because the clusters minimise the impact of change (change is usually confined to one subject area at a level). Entity model clustering is thus a simple solution to simple design objectives.

Entity relationship models become more stable and correct through the use of entity model clustering. Due to the structure of a clustered entity model and its maintenance properties, even very large models become easy to communicate, validate and maintain. Hence there will be improved user interaction and less chance of mistakes in diagrams, so more

confidence can be placed in a model as the basis for further development work. Additionally the resulting systems have more acceptance due to increased user participation in their development.

C7.3 ENABLES VIEWS OF MODELS TO BE PRODUCED

As described in appendix X2, one facet of producing a model is that it just represents a set of views of an area. However typically in an analysis these views are conglomerated into a single diagram. This causes communication problems when the single diagram is fed back to users. Entity model clustering helps by enabling views to be taken of diagrams (usually not at the lower-level) without altering the stability or meaning of the underlying model.

So a clustered entity model can be viewed at different levels of abstraction as desired, a view which can additionally be confined to an area of interest. This enables the people most concerned with the results of any analysis to comprehend the results more easily and completely, and also make better use of them.

Organisationally dependent views, eg. divisional and departmental, can be easily assembled using subject areas as fundamental building blocks. For example, a Purchasing

Department view can be taken of a corporate model, while leaving the underlying corporate model intact and using terms recognisable to Purchasing for the subject areas. Thus benefits of an often organisationally-oriented technique such as data-flow diagrams can be combined with those of an organisation-free technique such as entity relationship modelling without losing the inherent benefits of either; if anything the benefits are more powerful in combination. These organisationally dependent views can provide the basis for organisational change due to a better modelling and hence, comprehension of the way that information is used within and across organisational boundaries.

Entity model clustering was used in this way at Whitbread & Co Plc, where the organisational views taken have been used to validate the corporate model with Whitbread's divisions. Entity model clustering does not solve the problem of different views of a single entity type (see C9.2).

C7.4 EASES THE USE OF END-USER COMPUTING

Rapid technological change has provided end-users with the power to meet many of their own development needs using personal computing or prototyping facilities. This evolution of information systems imposes a new requirement on information system development, namely to provide an information management

service within and across organisational boundaries. One inhibiting factor is that system developers tend to express information in terms which reflect an underlying physical system whereas end-users deal in the real world. The interpretation of the need to provide an information service both within the business context and its subsequent translation to implementation terms is a gulf that has been recognised for end-user computing [MART, 84]; it is also key to the successful application of information system technology in general [ISAD, 84], [FLAV, 81].

After a system has been developed, any entity relationship diagrams on which that system is based are often just used for maintenance purposes. However it is possible to utilise the diagrams as an 'index' to the information content of an information system, whether computerised or manual. Some of the end-users of an information system will have taken part in the development process and would be conversant with the conventions used, conventions which are simple enough to allow easy training of other personnel.

The use of an entity relationship diagram in this manner would be facilitated by data dictionaries. However entity relationship diagrams that result from large or diverse developments are not suitable for end-user computing purposes due to failings in the way that entity relationship modelling is

currently applied. For example, dictionary support requires the user to know the exact name of an entity type or one of its many synonyms; names which are either buried in complexity on inaccessible diagrams, or not shown at all on a superficical model. Current dictionary output also has to be subsequently interpeted in the context of the business by use of an entity relationship diagram anyway. Thus the environment which would benefit most from end-user computing facilities faces the greatest difficulties in the dissemination of the basic information available.

Entity model clustering deals with the problem of large or diverse developments, so improving the use of end-user computing. For example, end-users can make better use of data dictionaries with a clustered entity model. The subject areas provide the context to an information request. An end-user does not have to know of the existence of an entity type before the request, but can be led to it through the succeeding levels of detail of a clustered entity model. At the outset of a request all an end-user needs to know is the rough location of information, eg., "something about products and customers roughly associated with ordering", without having to know the exact entity types concerned.

It should be noted that a data dictionary (DATAMANAGER) has been used to record the results of entity model clustering at Whitbread & Co Plc.

C7.5 USE OF ENTITY MODEL CLUSTERING IN INFORMATION STRATEGY PLANNING

To realise strategic, tactical and operational benefits of corporate data, its importance as a resource must be recognised [SIZER, 82]. The meeting of this need is hindered by problems in the application of entity relationship modelling which are primarily caused by difficulties in coping with medium to large volumes of information. This is precisely the situation which entity model clustering was developed to deal with, giving realisable benefits to any large organisation undertaking corporate data modelling.

Due to the reasons discussed previously, planning studies often result in superficial overview models containing little useful planning and development detail. This is beneficial for steering committee and management board reviews, but tends to be less useful for the planning activities necessary for developing information systems. For reasons of speed in planning it is useful to have little detail, but then more intuition has to be used. It is best to make planning decisions based on full information.

Similar problems occur with the necessary validation of the models produced by the development process. Entity model clustering allows models of sufficient detail for information

system planning and development activities without the complexity normally associated with large models.

Additionally, as entity model clustering aids the analysis of activities (see Chapter D), it will be of great value in the Function Analysis tasks of Information Strategy Planning.

Finally, one of the most important deliverables of an Information Strategy Planning project is a Business Systems Architecture defining business areas to be analysed and systems to be developed. Entity model clustering can provide valuable aid here, as I will now discuss.

C7.6 DEFINES BOUNDARIES

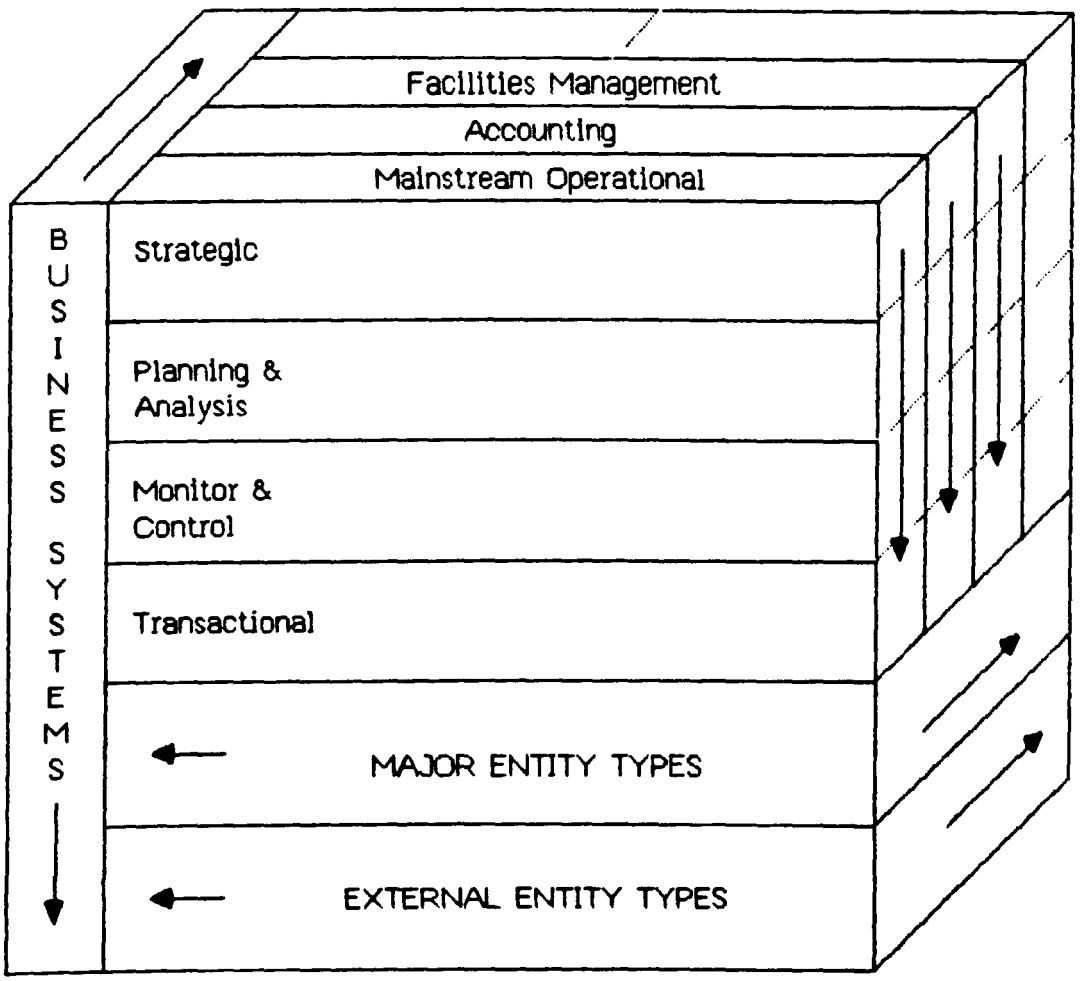
Given a clustered entity model, it is possible to use it to define boundaries of business areas to be analysed and business systems to be developed.

First it is important to briefly discuss 'extra-dimensional' activities, which are fully discussed in appendix X2.

We are accustomed to thinking about functions/processes as having dependencies in only two dimensions due to the restrictions of the paper we draw on. This affects the philosophy behind analysing and designing systems and can result in unnecesarily complex diagrams.

Certain functions are in fact "extra-dimensional" to the mainstream of operational functions. A prime example of this is Accounting, which potentially has dependencies on every other function - with consequential complexities in the corresponding entity type relationships. Another classic example is Planning functions which can also potentially require information from every other function and whose data can potentially be derived from every other piece of data.

So what exists is a multi-dimensional situation - not only are there the four different types of system (operational, monitoring and control, planning and analysis, and strategic), there are also systems which potentially transect every other system, and there are also the major entity types across all of this. Furthermore it is possible to isolate 'external' entity types, such as Public Authority/Government, which have a large effect on many parts of an organisation, but are totally outside the control of the organisation.


Facilities Management
Accounting
Mainstream Operational
BUSINESS SYSTEMS
Strategic
Planning & Analysis
Monitor & Control
Transactional
MAJOR ENTITY TYPES
EXTERNAL ENTITY TYPES

Many problems experienced in deciding business system/business area boundaries arise because this complex situation is reduced to two-dimensional diagrams and matrices.

Business area (ie. project) boundaries can be delineated through the use of a clustered entity model. An initial investigation will elicit broad areas of interest, the top-level subject areas. These can be made the focus of detailed investigations which are logically directed towards a result which bears a relationship to the developed models. For example, a project based around an ordering and distribution subject area will fully define that area and its surroundings; the choice of this area would be due to its forming a logical group in a clustered entity model rather than from being a department in an organisation, or from being a known major function of the organisation. The results from taking logical groups should be more stable and better directed than from other ways.

System boundaries may also be able to be delineated through use of a clustered entity model. The subject areas can be made the basis of information systems, with the interfaces pre-defined by the clustered entity model. The interfaces can be investigated to provide system dependencies for use in development planning. It must be emphasised that a clustered entity model would not be used in isolation to define projects and systems, but would be used in combination with techniques such as cluster analysis (Murtagh gives a good review of cluster analysis methods in

[MURT, 83], [MURT, 85]). However it is important to note that entity model clustering makes the use of these other techniques much easier as logical groups of data are already defined for them.

Subject databases, a.k.a. natural data stores, are collections of entity types which it is desireable to develop and use together [MART, 81]. The identification of subject databases and their interfaces is aided through entity model clustering. First-cut design of many subject databases can be gained from subject areas as these are clusters of related information all concerned with the relevant major entity types and all likely to be used in the same way. For example, a Customer subject area roughly maps to a Customer subject database as all the information relates to customers and will all be used in customer-oriented processing.

This discussion has applied to Information Strategy Planning. However it is perfectly possible to apply the results at the analysis level to isolate design areas (parts of a business area to be designed together) and to redefine business systems and subject databases for that business area. The technique and discussion are exactly the same except applied to a smaller part of an enterprise.

I will now discuss the steps needed to form boundaries of business areas. The same can be used to define design areas.

AREA BOUNDARY FORMATION:

C7.6.1 Extra-Dimensional Functions Subject Areas

Isolate extra-dimensional functions and their subject areas. These are typically characterised by a large number of relationships to many other areas and largely form 'sinks' in function dependency diagrams.

Extra-dimensional functions/subject areas should form business areas of their own, depending on the results of succeeding steps.

C7.6.2 Major Entity Types

The major entity types form the basis for a corporate data infrastructure. A business area should be defined which encompasses them and any subject areas and functions solely dependent on them. The analysis of this business area should be carried out before any other Business Area Analyses, and should be an input to them - a result from this Business Area Analysis is the backbone of corporate data. This can be extended to cover any external entity types that have an affect on a wide part of the organisation.

If this Business Area Analysis is not done, the major entity

types will be defined piecemeal from every other Business Area Analysis - with consequential data administration problems - though this is a valid approach.

C7.6.3 Form 'First-Cut' Mainstream Systems

The remaining functions and their subject areas can be considered as candidate business area/business systems - one per function.

C7.6.4 Form Business Areas/Business Systems

The set of business areas/systems can be manipulated if desired - large areas can be split into their component functions/subject areas and small areas can be grouped.

A few guidelines:

a) The bond between a function and the subject area it applies to is inviolate - whenever one appears in a business area, so must the other.

b) Major entity types related to a single subject area must be in the same business area.

c) Subject areas can be grouped according to relationships with each other, by logical horizons, or by relationship to

the same major entity type(s).

d) Lowest-level subject areas must not be split across business areas; ie. business areas should contain integral entity type abstractions.

e) Usual methods, eg. cluster analysis (see C8.1), can be used on the restricted set of functions and the subject areas:

First form a matrix of functions against subject areas. Fill in the cells with 'change' or 'enquire' actions. The subject areas should be listed in a matching order to the functions so that a 'change' diagonal is formed. For the best results, the lowest-level subject areas should be used.

Given this, it is then possible to use cluster analysis algorithms on the restricted matrix to group the subject areas.

An alternative is to find the recursive-transitive closure of the following process:

Take a subject area

Find its primary function (the function which most closely reflects it)

Find other subject areas which that function changes

Repeat with these subject areas.

The result is a group of functions which all change the same areas - a good basis for a business area.

It is also possible to do this process by starting with a function, finding the primary subject area, find other functions which change this area and repeat with these functions - the result is a group of subject areas all changed by the same functions. The results are not necessarily the same.

C7.6.5 Manipulate Results

Consider all the business areas/systems formed previously and see if any extra-dimensional areas can be grouped with the mainstream areas, and if any/all major entity types are best grouped with them.

Guidelines apply as for C7.6.4.

C7.7 AIDS AUTOMATION OF ENTITY RELATIONSHIP MODELLING

The automation of support for entity relationship diagrams is becoming more widespread, with prototypes (eg. chapter E) and even commercial products available. However with modern technology these suffer from even more acute problems than the traditional media due to the restrictions of the screen resolution, which is often the equivlent of an A4 sheet of

paper.

With automated diagrams, a complete diagram will not be able to be seen at one time. Even with very good resolution, there will be a point where things are so small that they will be incomprehensible. The alternative is to 'scroll' through a number of pages. The loss of context inherent in this process is bound to cause comprehension difficulties. Entity model clustering would allow complete diagrams to be viewed and the context of separate diagrams to be retained, thus easing restrictions and enabling entity relationship model automation to be more successful. Clustered entity model automation would consist of ordinary entity relationship model automation (as in chapter E) combined with that of movement through a data flow digram hierarchy.

This has actually been implemented on Excelerator [INTE, 86], a data flow diagramming tool which allows the explosion of any object to a lower level.

C8 COMPARISON OF ENTITY MODEL CLUSTERING WITH SIMILAR TECHNIQUES

Entity model clustering was novel as a technique in its original application of basic ideas that were identified for other areas of modelling. The technique has novelty in its entirety and in its detail as applied to entity relationship modelling.

When entity model clustering was developed I was not able to find any similar formal techniques. The closest was the use of cluster analysis algorithms. This is discussed below in C8.1. In fact it turns out that entity model clustering is a form of cluster analysis.

Many informal, intuitive methods of subsetting a diagram to improve communication have been used, but attempts were never made to link the subsets together, except in a grandoise corporate model covering large amounts of wall-space. One common method of subsetting was to identify broad groupings intuitively and to colour in a complete diagram to represent the subsets.

Recently I found a paper from Lockheed [GILB, 85] which described a method very similar to entity model clustering. I hasten to add that a small paper describing entity model clustering was printed 1 1/2 years previously [FEMI,85] and a larger paper describing entity model clustering [FEMI, 86] had

been accepted for publication a year before the similar technique was published, but had not yet appeared due to the long lead times of The Computer Journal. The technique itself had been discussed in open forum at a BCS Database Specialist Group meeting six months earlier.

This similar technique is briefly discussed in C8.2. I consider entity model clustering to be superior to this technique.

C8.1 GROUPING BY CLUSTER ANALYSIS

The following is an extract from the BAA Handbook published by James Martin Associates [JMA,86b], but which I wrote.

Cluster analysis enables us to investigate the affinities of objects to each other. This allows us to make decisions about how important it is to group objects together. The criteria used is that data which is used in broadly the same way is grouped together and that activities which use broadly the same data in the same way are grouped. The main benefit to cluster analysis is that it provides us with sensible groupings of objects to design together.

A cluster analysis algorithm is a method of grouping objects together. Four broad types of algorithm have been found to be useful: clustering by asociation, clustering by affinity of use, hierarchical analysis clustering and manual row/column moving.

The latter three all make use of a process/entity type matrix as the source data. In the first three methods a separate clustering is carried out for the entity types and the processes.

a) Clustering by Association

There are two techniques for clustering by association, entity model clustering (the subject of this chapter) and process dependency clustering. These both consider how to group objects together by just considering the association between objects.

Entity model clustering considers grouping by entity type relationships. Process dependency clustering considers grouping by process dependencies.

On a process dependency diagram, we can determine all the processes that another process is dependent on, and which are dependent on it. Ideally in a business system the dependencies outside the system will be minimized, but we are unlikely to be able to totally eliminate such 'interface' dependencies. Where interface dependencies occur we wish to minimise the cost of implementing the dependencies. This is the purpose of process dependency clustering. We require to group together those selected processes which have a high degree of interdependency. Additionally any dependencies outside of the group should be optional and just require a single use of information view.

b) Clustering by Affinity

The affinity of an object for another object considers the similarities of the objects, or how closely associated they are. This can be assigned a value, the higher the value the closer the affinity.

As an example, for two entity types E1 and E2, we can define the affinity of E1 to E2 to be:

$$\text{Affinity of E1 to E2} = \frac{\text{Number of Processes using both E1 and E2}}{\text{Number of Processes using E1.}}$$

For any pair of entity types we can define an 'average affinity' as:

1/2 [Affinity of E1 to E2 + Affinity of E2 to E1]

Similarly we can define the affinity between two processes, P1 and P2.

$$\text{Affinity of P1 to P2} = \frac{\text{Number of Entity Types used by both P1 and P2}}{\text{Number of Entity Types used by P1}}$$

and an average affinity as:

1/2 [Affinity of P1 to P2 + Affinity of P2 to P1]

c) Hierarchical Cluster Analysis

One of the most successful methods of cluster analysis is the use of "hierarchical clustering" algorithms, of which "Wards Algorithm" has produced good results. These algorithms calculate the dissimilarities between objects (as opposed to affinities), thus enabling the clustering of similarities. The clusters formed in this way are then clustered themselves until a single group is formed. Murtagh [MURT, 83], [MURT, 85] has surveyed most of these algorithms.

d) Manual Clustering

When clustering we need to form groups of entity types and processes so that the usage of entity types is fairly self-contained. However, most important is that all 'establish' actions are in a defined group. It is possible to manually cluster a matrix, regrouping all 'establish' uses of an entity type together, and grouping all entity types created by the same processes together.

To enable a higher degree of dissimilarity or affinity to be obtained, we can make use of weightings to distinguish type of use. We can give a different weighting to an 'establish' use to a simple 'select' use. So if we wish to give the clustering of 'establishes' more importance than selects, we would give 'establish' uses a higher weighting (hence producing a higher

dissimilarity value).

When we have the results of a mathematical clustering (ie. either from b) or c) we need to be able to interpret them in a useful manner. One way is to construct a dendrogram. A dendrogram is a tree-like structure where the leaves are the clustered objects and the root is the least similar cluster.

In a dendrogram the objects are listed in affinity order, so that objects and clusters with the highest degree of similarity are next to each other. The objects and resulting clusters are joined by 'square arcs' with the length of the arc determined by the degree of similarity between them.

At the end of mathematical clustering we know the degree of affinity or dissimilarity between every object. The results are then analysed by taking a cut-off value above, or below, which all objects are considered to from a reasonable grouping, ie. which have enough similarity to be considered as a worthwhile grouping for design.

So, as cluster anlysis is a method of grouping objects together, entity model clustering is a subset of it, and in fact is a cluster analysis algorithm. In the handbook I suggest that more than one method be used for the best results.

Note, however, that cluster analysis does not lead to a

clustered entity model. Cluster analysis is just one means of forming the clusters. The main benefits discussed in C7 come from the clustered entity model itself, entity model clustering is just a means to an end.

C8.2 LOCKHEED TECHNIQUE

The only technique I have found which resembles entity model clustering is one developed by Lockheed in California [GILB,85]. They were forced to develop some method to help control a large (>250 entity type) entity relationship model.

The method they chose bears a similarity to entity model clustering in that they structure by showing all the entity types which apply to, what they call, a given subject which appears to be equivalent to a subject entity type (see C3.2). There is one page per subject. This would be equivalent to producing a page per major entity type showing all the entity types that applied to it. Entity model clustering produces a subject area diagram which is the logical horizon of some minor entity types, so is in effect the decomposition of the relationships between major entity types. Where the logical horizon encompasses a single major entity type the result is the same.

I believe my approach to be better because it shows a more natural grouping of data; people tend to think in connections

between objects, as opposed to all the data that pertains to a given object. Additionally, my approach leads to largely 'functional' groups, where it is often easy to find the users who deal with that data; they are likely to form a coherent group to communicate to. It is not nearly so easy to group together all those people who deal with a given major entity type. For example, if all the people who deal with a Product in an enterprise were grouped together to validate a diagram there would be few people left working.

The pulling together of these views at the same time is also not a task I would like to take on. To summarize, I believe the function-oriented result is more intuitive and easier to manage then the data-oriented result. It is telling that Gilberg says that it is the data analysts who use the Lockheed technique, not the business analysts (the people who communicate with the users).

Additionally the Lockheed method does not form a hierarchy, it is a structuring on a single level and there is no levelling to lead to the detail. This is a vital part of a clustered entity model and produces many of its benefits.

To conclude, while Lockheed have recognised the need for a structuring technique and produce similar results to entity model clustering, the results are not as useful as a clustered entity model, nor as easy to use.

C9 FURTHER RESEARCH IN ENTITY MODEL CLUSTERING

Entity model clustering is basically quite a simple technique and has been fully researched. Some of its implications were not researched though, just identified and require further research.

C9.1 BLUEPRINTING MODELS

There is a well known adage that "there is nothing new under the sun". This applies very well to organisations, which tend to be very similar in similar situations. For example, Purchasing tends to be more-or-less the same in most organisations and Sales tends to be similar to Purchasing in most organisations as well.

Of course this is the basis of application packages, which are standard solutions. The problem with application packages is that they require tailoring to a particular situation, which often reduces their cost-effectiveness.

It would be very nice to be able to produce 'blueprint' entity relationship models that only needed to be tailored to a particular situation, especially if a system can be generated

easily from them. The problem is to ensure that a blueprint model matches a given organisation; it is often the exceptions to standard situations that are the most important aspects of a system. If a complete model is used there are all the problems of being able to comprehend the implications of the model.

A clustered entity model should make this much easier due to its communication properties. The parts of a model which are pertinent to a user can be shown and the rest ignored at that time. This needs to be attempted.

C9.2 ENTITY TYPE VIEWS

As discussed in C7.3, subject area views can be taken of a model while leaving its meaning intact. However views could not be taken of the entity types themselves. This would be most important for major entity types, which would be the area of most conflict in an enterprise because they affect so much of it.

I suspect that a similar concept to entity model clustering can be applied to individual entity types (ie. applied to an entity type's attributes) to form views of the entity types. When combined with the full entity model clustering, it should enable true user or purpose views to be taken of an entity relationship model without affecting the base information in any way. It would also help with the blueprinting discussed in C9.1.

C9.3 ENTITY MODEL CLUSTERING AND DECISION SUPPORT SYSTEMS

As described in B7.4, entity relationship diagrams can be used to represent facts for an expert system. If complex areas are analysed in this way, it will be necessary to cluster the diagram to enable it to be communicated. This has not been attempted as yet. It would also be interesting to investigate the effect entity model clustering has on modelling for Decision Support Systems. One aspect of this returns to entity type views as discussed in C9.2. It is occasionally proposed that Decision Support Systems use unstructured data, but it is probable that Decision Support Systems use as structured data as anything else, it is just a totally different **view** of that data to the rest of an enterprise. If entity type and subject area views of corporate data can be taken, it should be possible to isolate the sources of data for a Decision Support System.

C9.4 AUTOMATING ENTITY MODEL CLUSTERING

As of writing, Whitbread & Co Plc are looking at the implications of automating entity model clustering. Due to restrictions of the environment they are working in, entity model clustering has had to be adapted to help them automate. This is why this aspect is discussed here as opposed to the main body of this chapter. I am not yet convinced that the route they are taking is the best.

A new definition of major entity type is being adopted: an entity type in more that one logical horizon. This helps automation because fewer decisions need to be taken and no boundary conditions ever occur. With this definition the basic algorithm can be used and forms groups with no relationships to any other group. These groups are then formed into subject areas by clustering those groups by minimising the connectivity of the areas. This is an easily automatable algorithm but doesn't necessarily produce human-friendly results. I believe it would be better to stick with the original concept and employ a Decision Support System to help the clustering.

Whitbread believe in the concept of entity model clustering enough to resource this effort, they just wish the formation process to be as machine-bound as possible. I believe further research would be fruitful in achieving this without changing the original concept.

CHAPTER D
PARALLELISM BETWEEN DATA AND ACTIVITY

CHAPTER D PARALLELISM BETWEEN DATA AND ACTIVITY

D1 INTRODUCTION TO THIS CHAPTER

The main research, into action modelling and into entity model clustering, was in different areas of modelling, modelling activity and modelling data respectively. However a theme which continualy cropped up was that the basic concepts and considerations were very closely related and bore a startling similarity to each other. This led to an investigation of this similarity, which resulted in a 'theory' that data and activity can be modelled in exactly the same way. More formally:

> "Every data structuring concept has an equivalent activity structuring concept with the same meaning and vice-versa."

For example, the concept of optionality in data modelling is exactly the same in activity modelling.

I do not necessarily consider activity and data to be the same thing, though it is possible to think of activity as data; a process is of significance to an enterprise so could be treated as an entity type. This is so for all activity. However there is data which cannot be 'executed', eg., in no sense can a Product be executed. So the basic difference between activity

and data is that activity can be executed while basic data never can. This is completely consistent with the Artificial Intelligence languages such as LISP [MAEHL, 82] and Prolog [CLME, 81] where there is no ready distinction between data and activity other than the fact that activity can be executed.

So there need to be activity models and data models. It is highly desireable if these can be modelled in the same fashion for simplicity of method. It is also intellectually pleasing to be able to find a pattern that brings together strands which are often considered to be separate. Indeed, as mentioned in appendix X1, it is an aim of Information Engineering to achieve consistency in this area. However it is important to note that the research will apply to any methodology, not just Information Engineering which happens to be a useful context for the description of the work.

I claim that introducing action modelling and entity model clustering has enabled the complete symmetry to be found. A meta-model has been produced of Information Engineering's 'Conceptual' objects including my research; it is shown in figure D1.1. This model will be one basis for 'proving' the symmetry in the following discussion. Figure D1.2 shows the situation before my research was introduced to Information Engineering. The differences are quite revealing.

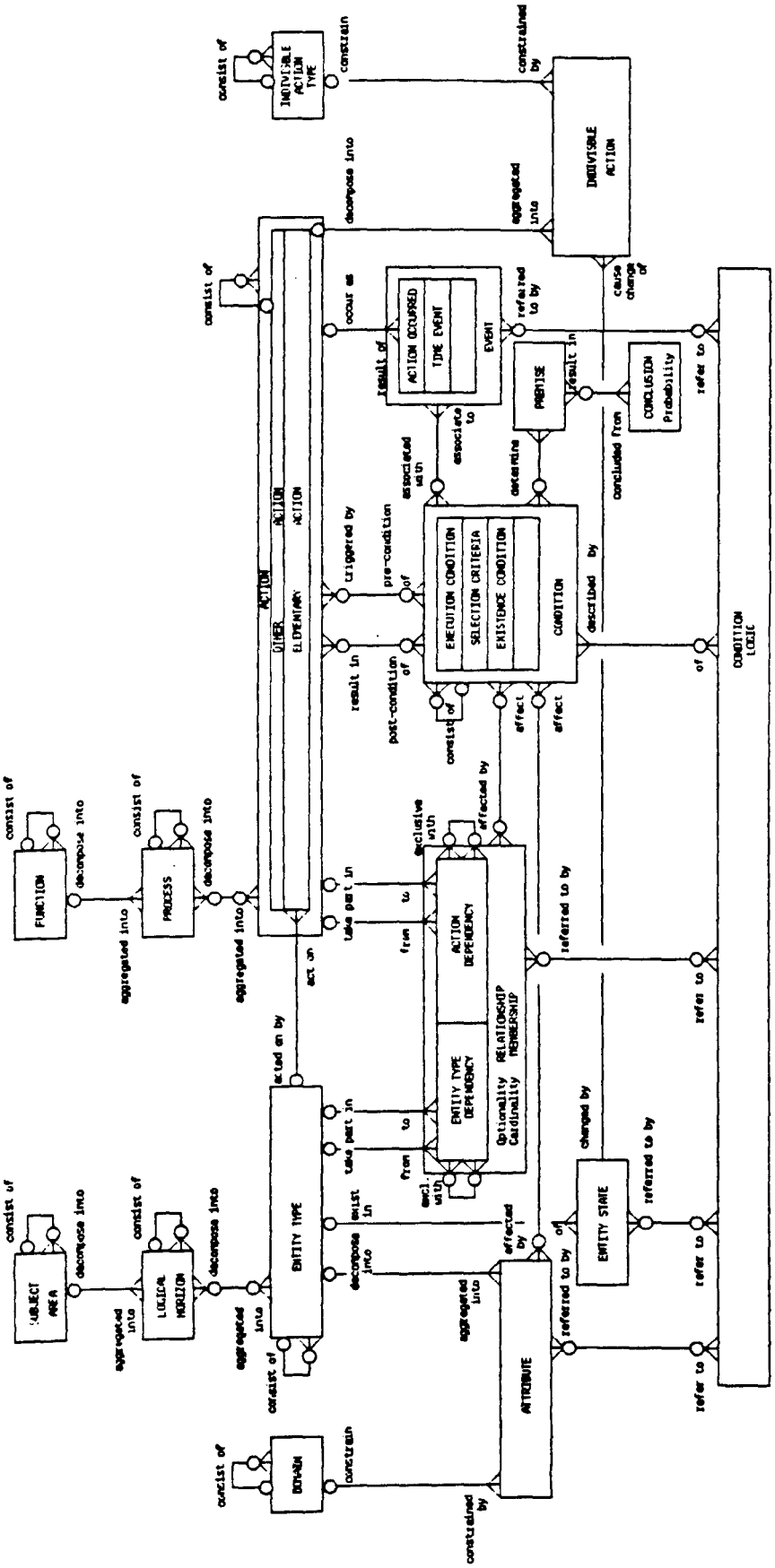


Figure D1.1 Meta-Model of Information Engineering Conceptual Objects

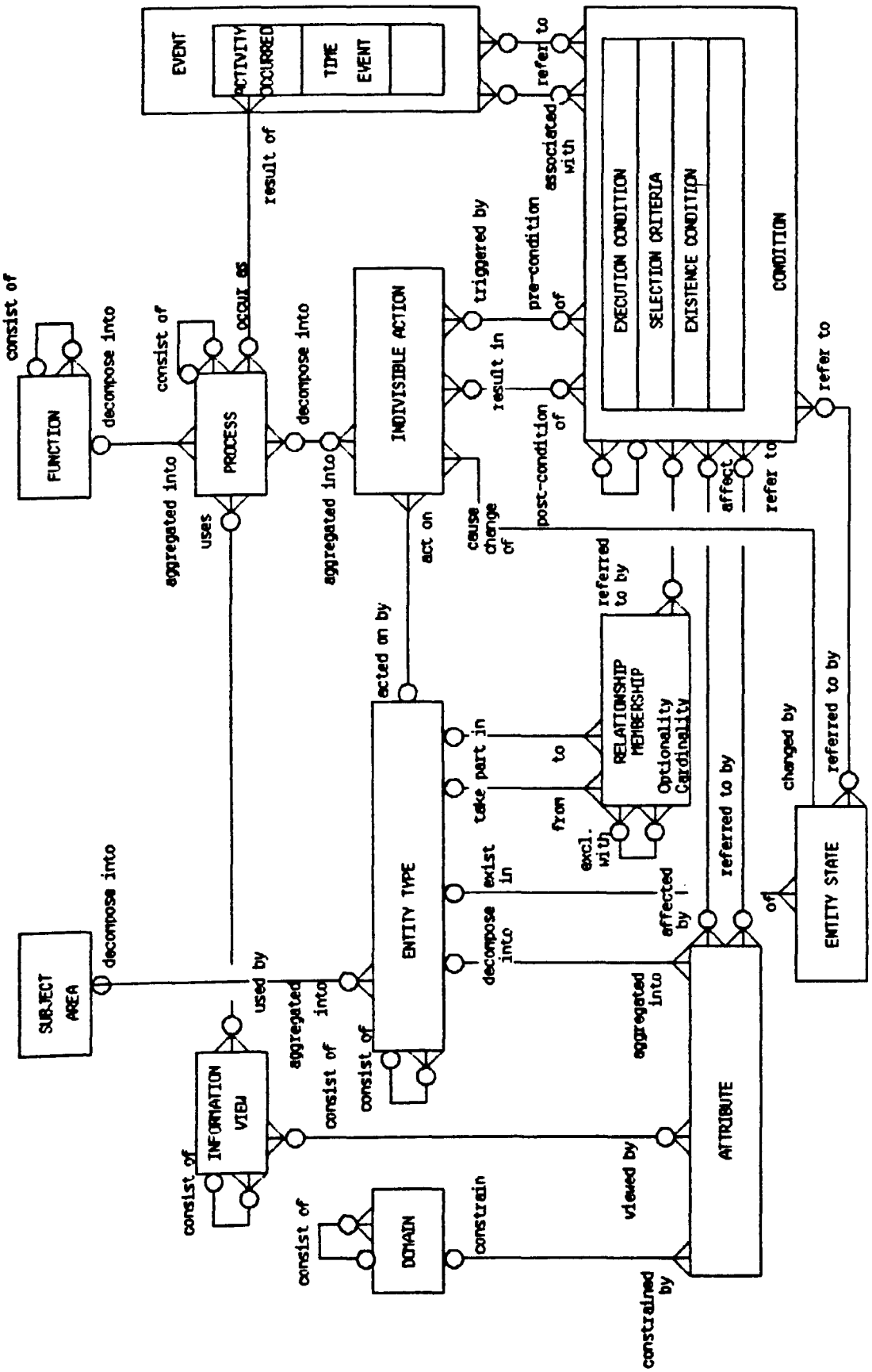


Figure D1.2 Meta-Model of Information Engineering's Conceptual Objects Before My Research

The symmetry has two main aspects: symmetry of objects modelled and symmetry of association. These two types of symmetry are explored first in the following and any affect on action modelling considered. After this some further research in this area is discussed.

Note that this theory has not been used in practise so no details of use are given. Also, as it is not a technique, just a theory, there are no actual examples to be discussed.

D2 OBJECT SYMMETRY

The basic 'Theorem' of object symmetry is that:

> "Every data object has an equivalent activity object and vice-versa".

A 'proof' of this can be seen if figure D1.1 is inspected.

The equivalence in table format is:

DATA OBJECT	ACTIVITY OBJECT
Subject Area	Function
Logical Horizon	Process
Entity Type	Elementary Action
Attribute	Indivisible Action [eg. ESTABLISH, UPDATE]
Domain	Type of Indivisible Action

D2.1 SUBJECT AREAS AND FUNCTIONS

Subject areas and functions deal with broad groupings of an organisation in data and activity terms respectively. Functions are often described as being 'activities whose occurrences cannot be envisaged', eg., it is not easy to imagine a Stock Management function executing, it is the processes within it which execute.

Similarly, occurrences of subject areas are not easily envisaged, eg., a Purchasing Subject area does not have an easy occurrence representation. Subject areas and functions are very closely related. I consider that subject areas and functions are either the same thing, or subject areas are just the data needed to support the functions. The latter is the most likely as the following discussion shows.

An organisation has its existence based around a set of entity types which are fundamental to the organisation and without which the organisation would cease to function. These are the major entity types. The set can change over time as the organisation mutates, but there will always be a set. Examples for a manufacturing company are: Product, Raw Material, Equipment, Employee, Customer and Supplier. Without any of these the company could not manufacture.

Given the basic major entity types, an organisation must do something to them - the functions of the organisation. The functions depend on the major entity types, but are determined by the nature of the organisation. The functions are mainly concerned either with forming interactions between the major entity types (eg., Selling links Product and Customer), or with the administration of the major entity types (eg. Supplier, Administration, Employee Education).

There may be functions concerned with using the major entity types and their associated information at planning and analysis, or strategic levels, eg., Territory Planning, Product Planning, Manpower Estimation. There may also be functions that plan other functions, eg., Purchasing Planning. Though it is often hard to distinguish between planning the data and planning the function, they are not always the same thing.

Depending on the level of a function, every function will have a subject area to support it. These will contain the data required by the function. A function can use other data, and its data may be used by other functions, but there will always be a primary area of data for a function. This function is known as the Subject area's **primary function.** For example, there could be a purchasing subject area to back-up a purchasing function, or a Production subject area to support a production function. However, it is only useful to consider areas for the higher-level functions - possibly only the first three levels - and it is not useful to consider areas for planning-like functions (a 'planning and strategic' area may be required, though the entity types would be obscure).

So to find functions, investigate the interactions between each major entity type and also consider if there are functions specifically to support a major entity type. These functions are probably secondary-level functions and can be grouped into 'top' level functions by considering their use of the major entity types.

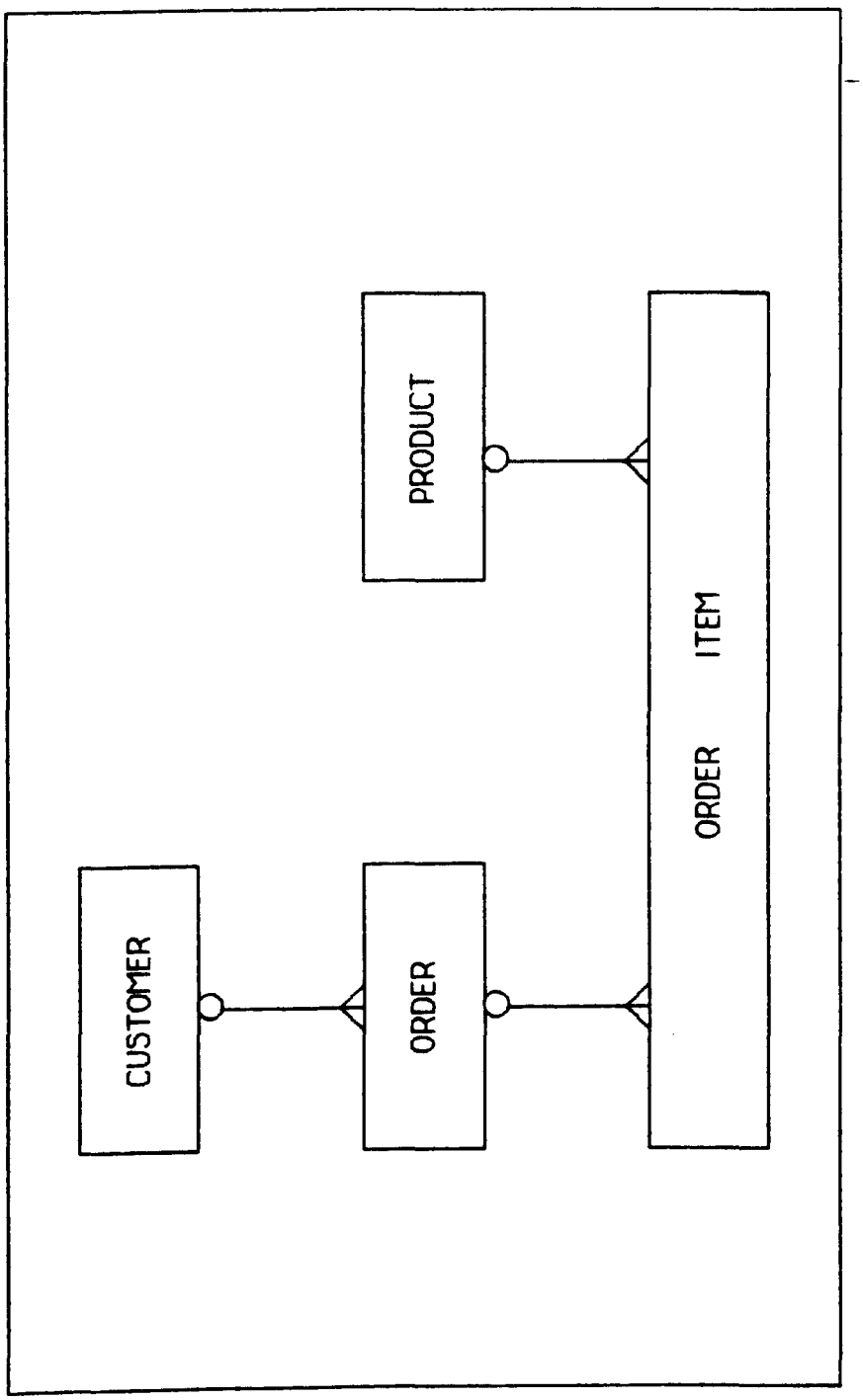
Next produce a subject area for each 'top' level function and a lower-level subject area for every corresponding secondary-level function. Add functions for planning, etc. and decide if areas are needed to support them. Finally the interactions between subject areas can be found as at present, however they can also be derived from the dependencies between the areas' primary functions.

Really all I am saying here is that every function must have a subject area to support it, but some subject areas may support more than one function.

D2.2 LOGICAL HORIZONS AND PROCESSES

Processes are convenient units of processing for an organisation; they form units-of-work and broad meaningful groupings of such units. Processes are not the basic building blocks of activity, but at the elementary level form recogniseable units of consistency.

Logical horizons also form convenient units, but in this case it is units of data. They are broad groupings, which tend to be meaningful. For example:


PRODUCT
ORDER ITEM
CUSTOMER
ORDER

The logical horizon of Order Item approximates an "Order Form", which is the basic unit of ordering in an enterprise. Whenever an order is taken all the objects within the horizon would be affected in same way.

There are few processes which act on a single entity type, it is much more likely that a process would use a logical horizon at some level. It is also likely that the higher the level of a process, the higher the level of the logical horizon it uses (as with subject areas and functions).

As an example, the Take Order process would use the complete Order Item logical horizon. If there is a Take Order Item process as part of Take Order, it would deal just with the immediate horizon of Order Item (including Order and Product but not Customer).

So processes and logical horizons both deal with groupings of their objects, entity types and actions respectively, and the units of data used by processes are logical horizons.

D2.3 ENTITY TYPES AND ACTIONS

Entity types are basic units of data that are recogniseable to users as a coherent whole, yet are non-redundant in that they are at least in Third Normal Form [CODD,70]. Entity types are the highest-level of data abstraction for which this can be

said.

Actions are basic units of processing, again which are recognisable to users as a coherent whole. Elementary actions are non-redundant. Elementary actions by definition act on a single entity (note not the entity type) and are a 'sequence' of processing on this entity. As there is a sequence, all parts of the elementary action must be dependent (any non-dependent part would be 'in-parallel'). Additionally, as it is on a single entity there will be no repetition of processing if Jackson's structuring rules are applied [JACK,75] (as the entity is non-redundant, there is no repetition, hence actions on it will not repeat). So an elementary action will be non-redundant. This property falls out because of the definition of an elementary action, which is such as to achieve non-redundancy, but also because it is acting on a non-redundant object - the entity.

Again the symmetry between the objects can be seen here.

D2.4 ATTRIBUTES AND INDIVISIBLE ACTIONS, DOMAINS AND TYPES OF INDIVISIBLE ACTIONS

Attributes are the lowest-level parts of an entity type. Indivisible actions are the lowest-level parts of an action.

As entity types and actions are symmetrical, I contend that attributes and indivisible actions are symmetrical.

Domains give the range of allowable values that an attribute can have, in effect they constrain an attribute to a set of values. If two attributes belong in the same domain then they are comparable, but not otherwise.

Likewise, the 'Type of Indivisible Action' is a constraint on an Indivisible Action, in that an indivisible action can only be one of the values defined by the 'type'.

So again domains and the 'types' are achieving the same end and constrain symmetrical objects, hence they can be considered symmetrical.

D2.5 OTHER OBJECTS

D2.5.1 Conditions etc.

Conditions affect all objects in an equivalent manner. There is no distinction in the way that a condition affects a data object and an activity object, mainly because conditions are basically enquiries on the state of an enterprise, normally to see if there is an occurrence of an object.

As conditions affect data and activity equally, so should premises and probabilities. I cannot find a counter-example to disprove this.

D2.5.2 Associations

Again these are symmetrical if not identical, but are considered in greater depth in D3.

D2.5.3 Entity States

Entity states are data objects pure and simple. The life of an entity is governed by the allowable states, it can pass through, these being defined for the entity type. They have great importance in data modelling as they govern the allowable transformations of an entity in any resulting system.

It is possible to define the states of an action. These tend to be simple and have no current identified use in designed systems (though this may change with more advanced architectures such as dataflows). The basic action states are NULL- STARTABLE- EXECUTING- WAITING (for data or reply)- COMPLETE, and are the same for all actions.

So entity states have an equivalent activity modelling concept, just one that it is not particularly useful at the present time.

D2.5.4 Events

Events were described in B3.3. They are useful for representing when a behaviour pattern should occur. As such they act as 'entry points' to the behaviour pattern and 'trigger' actions.

There is an equivalent data modelling concept which is called 'entry point'. These are not normally identified as a conceptual object, but are identified in design where they describe the allowable means of access to a record type.

So events have an equivalent data modelling concept, just one that is not particularly useful to analysis.

D3 ASSOCIATION SYMMETRY

The basic theorem of association symmetry is:

> "Every data association concept has an equivalent activity association concept and vice-versa"

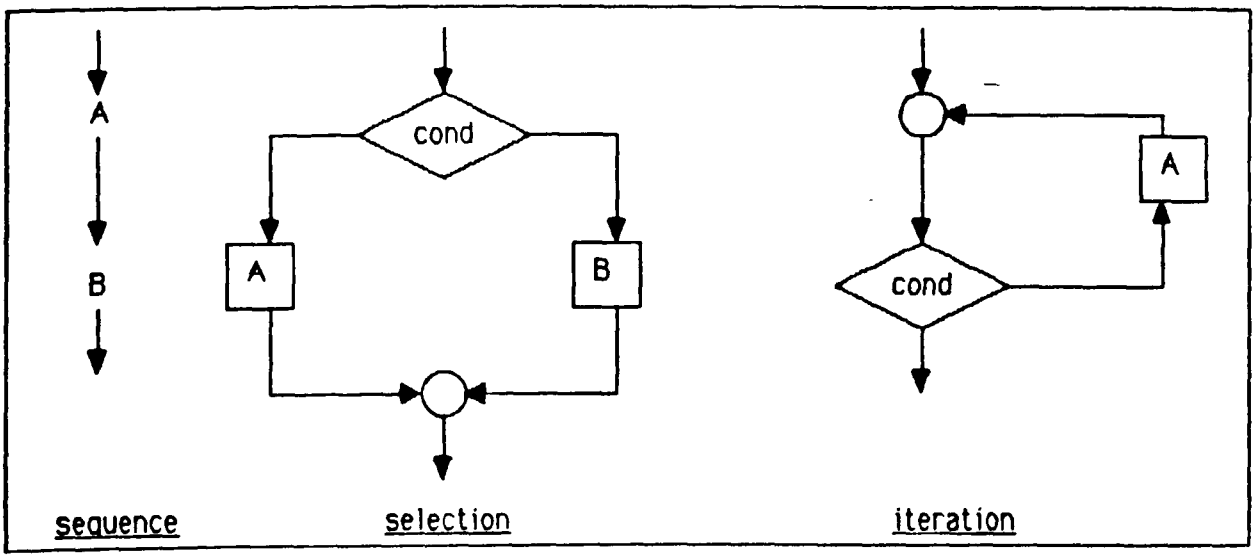
(I have another private theorem which is that data and activity association are identical in every respect; as yet I have not been able to prove this.)

Abstraction is one aspect of this, as described in appendix X1. As this discussion shows, where objects are symmetrical the abstractions that apply to them are also symmetrical.

It is also possible to define symmetry between the other association constructs. (The following discussion is taken from a co-authored paper [FMM, 86], which is based on these theorems.)

D3.1 ASSOCIATIONS IN ACTIVITIES

Bohm and Jacopini [BOJA,66] proved that all program logic could be expressed using only three basic control structures, called sequence, selection and iteration:



These could be nested to any depth, so A and B above could themselves be one of the three control structures. The program was seen as a hierarchy of control structures with leaf nodes which were processing statements performing, for example, assignment or arithmetic.

One of the first serious approaches to program design, developed by Michael Jackson [JACK,75], was based on these ideas and the observation that many data structures can be described in terms of the same three basic structures: sequence, selection and iteration. In this approach the data structures were drawn as hierarchies and the program structure was inferred from them.

Jackson makes no attempt to attach significance to the idea of sequence other than through data structure and proves the fact that sequence is often arbitrary. For example:

```
MOVE A TO B
MOVE C TO D
```

can be coded in any order without affecting the result. It is only the nature of current computer programming which makes us choose a sequence. The two statements below are different:

```
MOVE A TO B
ADD 1 TO B
```

Reversing the sequence of these two would lead to a different result because the second depends on the first for a value of B. There is a data dependency. This gives a more precise idea of sequence. The previous example illustrates the idea of parallelism where the two statements could be executed together if there were a computer system capable of it.

It is impossible to build adequate models without the notions of sequence based on association (dependency) and parallelism.

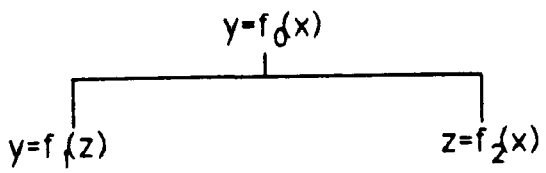
An approach to program design based on dependency and data flow was evolved by Yourdon and Constantine [YOCO,75]. In this approach, the program function was decomposed into sub-functions

using data flow diagrams. The diagrams were then transformed into a module hierarchy. This approach tended to overcome a criticism of the Jackson approach that it was not until the leaf nodes were reached that any meaning was evident. The Yourdon and Constantine approach resulted in modules whose function, inputs, and outputs were quite apparent. On the other hand the transformation of data flow diagrams to a module hierarchy is not rigorous and no guidance is given on the design of the modules' own internal structure.

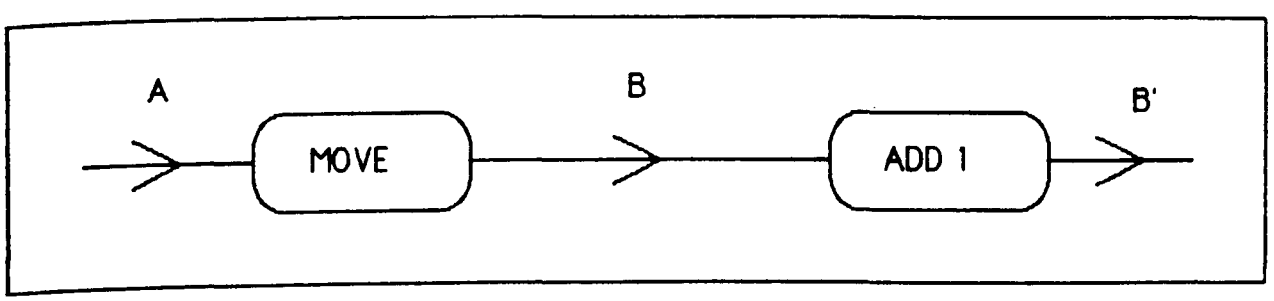
The ideas above were reconciled by Hamilton and Zeldin [HAZE,75], [HAZE,79a], [HAZE,79b] in the HOS (Higher Order Software) approach. In this the relationship between control structures and association was detailed.

A system is described as a hierarchy of mathematical functions, with inputs and outputs. There are three primitive structures: JOIN, OR and INCLUDE corresponding to sequence, selection and parallelism respectively. Iteration can be constructed from a combination of selection and recursion (or invocation of a activity by one of its sub-activities). The three primitives are:

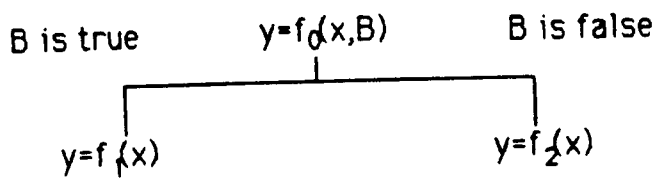
a) JOIN (sequence)



f_0 is composed of f_1 and f_2 where the output from f_2 is the input to f_1. Using data flow diagram conventions we can represent our previous example of sequence as follows:

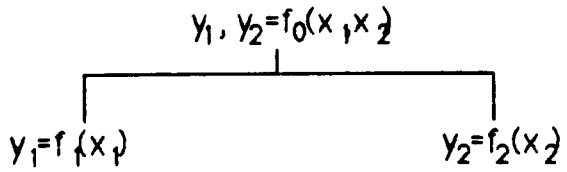


b) OR (selection)

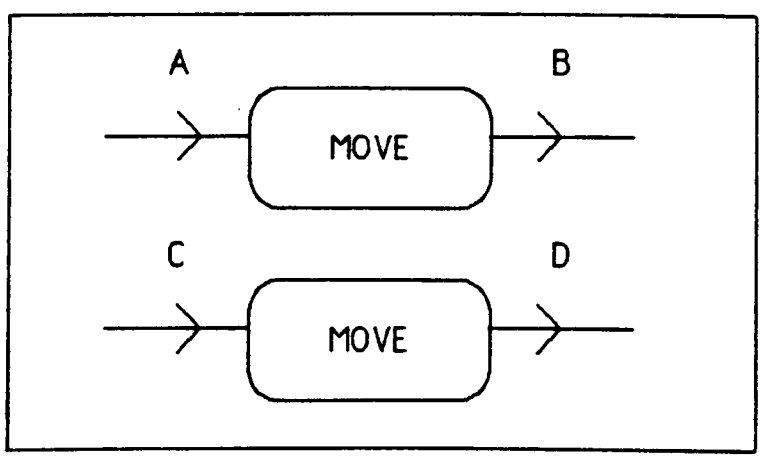


An execution of f_0 consists either of an execution of f_1 or of f_2 depending upon whether B (previously evaluated) is true or false.

c) INCLUDE (parallelism)



in this case f_0 consists of an execution of f_1 and an execution of f_2. In data flow terms again, the earlier example of parallelism appears as follows:

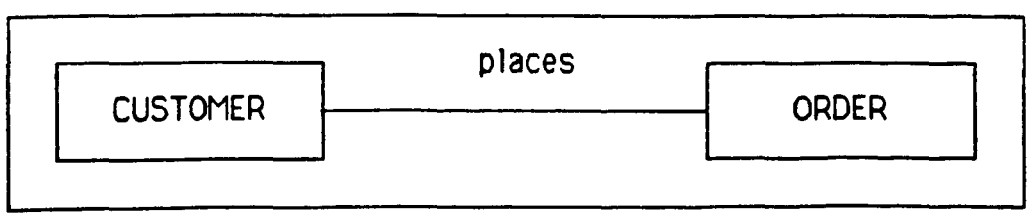


These ideas are of significance at every level of modelling.

D3.2 ASSOCIATIONS IN DATA

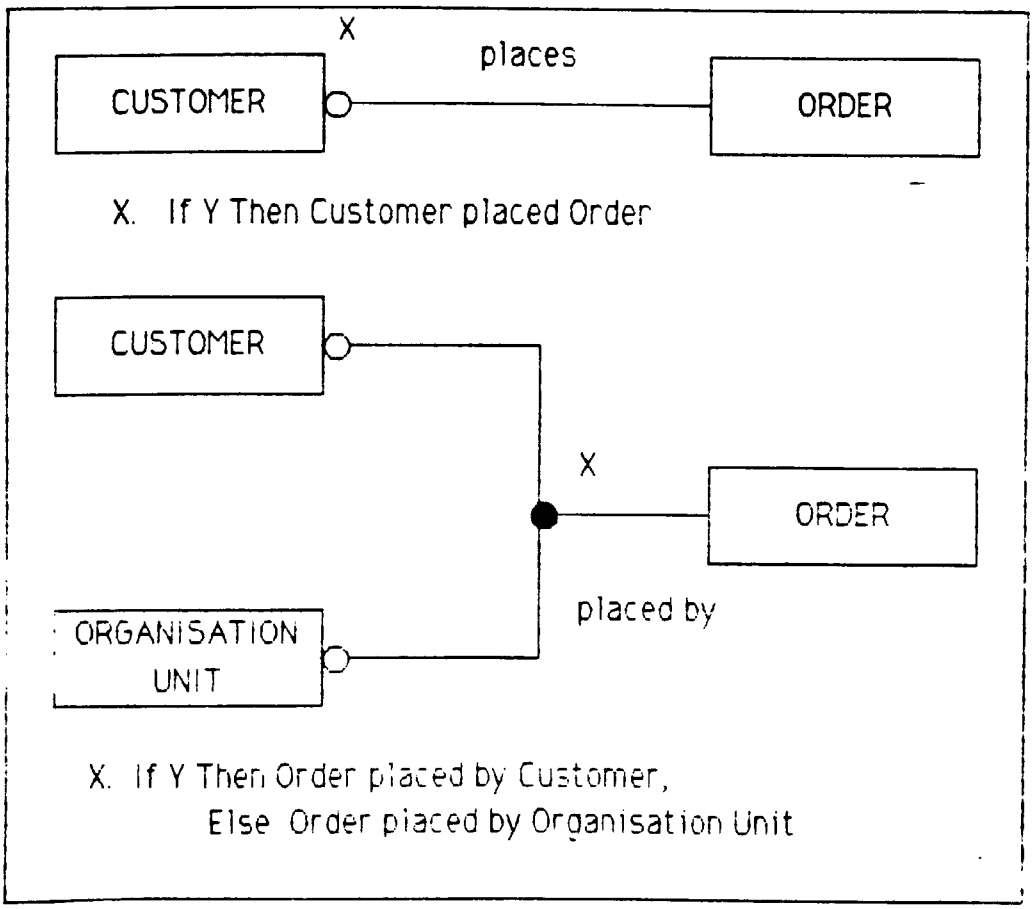
As we will show, it is possible to use primitives to explicitly describe the structure of data. They are based on the concepts previously described for associating activites. This is no surprise as we have been implicitly using these concepts for years in discussing the structure of data, for example in Jackson structured programming and entity relationship diagrams. Consider the case of entity relationship diagrams.

Any two entity types can be associated in any number of ways. Only some of these ways are of use and we identify these as relationships between the entity types. For example, if a customer places an order then the CUSTOMER and ORDER entity types are related by a 'places' relationship. Given the three primitives sequence, parallel and selection, a relationship corresponds exactly to sequence, as in:

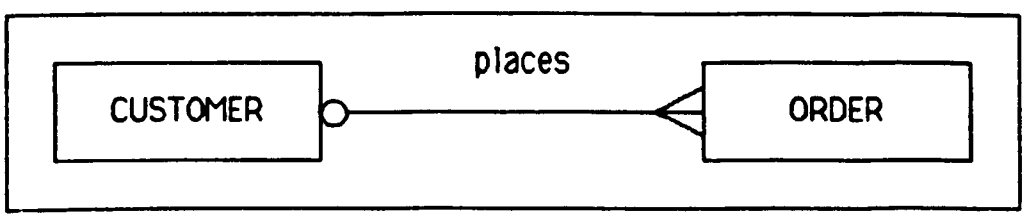


Hence two entity types are 'parallel' if there is no direct relationship, as in:

We know that relationships do not have to hold between entity types (for example, there is a relationship between Customers and Orders, but we often require to know about Customers who have no Orders, (eg. prospects)). This is called 'optionality' and is equivalent to selection, as there will be some condition to determine if the relationship holds between any particular Customer and Order, for example:



Finally, we also know that there is iteration in the relationship between two entity types; for example one Customer can place a number of Orders, therefore this is a repeated relationship for the Customer, as in:



D4 ACTION MODELLING REVISITED

After the theorems were recognised, it was found that action modelling could be made more rigorous. This section describes how. (Note that there was no noticeable affect on entity model clustering.)

It is possible to consider activity at various different levels of decomposition, all the way down to activities on 'sub-atomic particles'. Of course the latter is not particularly useful to us in developing information systems. Another, more practical, low level of activity is activity on attributes of entity types. This is in fact the lowest useful level of activity, but is rarely communicable. However this level can provide a set of axiomatic activities to abstract from, or to decompose to. If using some method such as HOS, it should be possible to form a decomposition down to this level and to prove correctness of decomposition. However one of the problems of this is proving correctness of requirements. There is little point in having a correct activity that does not satisfy some business requirement.

It is necessary to start from some validatable level and to decompose from there. Elementary processes are the lowest level of activity where consistency and completeness of the enterprise can be guaranteed on completion of an execution. An elementary

process is the level where informal decomposition normally ceases. However, for action modelling this is the level we will start at, mainly because the inputs, outputs and required transformations of an elementary process are verifiable with users.

As described in chapter B, any elementary process can be broken up into its constituent actions. There are dependencies between actions showing the selection, sequence and iteration of the actions. Parallelism is a default in this case - any two actions which have no necessary sequence can occur in parallel. As these primitives are the same as those employed in HOS, the decomposition of an elementary process into its actions is provably consistent with higher levels. This level of activity should still be validatable for consistency with business requirements, as recognisable business activities are still being considered.

Each action can then be considered in turn. If it is a single transformation of an entity type then it can be broken down into actions on attributes (the lowest level) and the decomposition proved correct with respect to the higher-level. If it is a sequence of transformations of a single entity type then it can be decomposed into a sequence on a single entity and then individual transformations of the entity; again this decomposition can be proved consistent because only sequence and selection can be involved. If an action is a sequence of

transformations of entity types in a logical horizon then it needs to be decomposed to a sequence of transformations on a lower-level of logical horizon or on individual entity types, and so on.

An important point here is that we have supplemented the concepts of consistency proving in HOS, with the systematic matching of the structure of activity to the structure of data in a top-down provable manner. This is made easier by the recognition that the structure of activity and of data can be modelled in the same fashion.

Above and including elementary processes we can verify the correctness of processing requirements within the business. However the more detailed the requirement, the harder it is to verify correctness. The constituents of the decomposition at elementary process level are generally determined by business rules, policy and common practise. This also applies to the immediate decomposition of an elementary process because recognisable business activities are still being modelled. Decomposition below this level generally involves discussion of detailed logic, which is rarely considered by the business and is often more concerned with details of implementation than with requirements.

With the concepts discussed here we can show that the output from a given level matches that from the next higher level and

that the data is processed according to the integrity rules the business requires. However we must check that the horizontal structure of inputs and outputs on a given level of decomposition match the business requirements. This can be by induction (ie. by verifying them at a level and trusting that the consistent decomposition mechanism continues to ensure that the business requirements will be met), in which case the lower the level of requirements correctness proving and the greater the amount of decomposition consistency proving employed, the greater the chance of producing programs which correctly meet business requirements first time.

D5 FURTHER RESEARCH

This chapter brought together the main strands of this research project. As this part was not the mainstream work there are loose ends, especially in the implications of the theory.

D5.1 ATOMIC DATA MODELLING

One interesting aspect which was developed internally at JMA with my collaboration is that it is possible to take the previous discussion and apply it to predicate calculus. Predicate calculus is concerned with modelling activities on data. If the symmetry theory is followed through there should be an equivalent set of concepts for modelling data. The discussion in appendix X3 is concerned with this. The further research here is to validate it and to examine its implications.

D5.2 ACTION STABILITY

Actions have a very close association with entity types; this is axiomatic to action modelling. Due to this, designs based on the results of action modelling should be as stable as the associated data. Thus the actual operations (the mapping of actions into design components, cf. entity types and record types) on a database should remain fairly static, whereas the

grouping of these operations into transactions can change as often as required with little affect on the database or even on the operations themselves. So in information system design the same amount of effort can be expended in the design of a network of operations as is often expended on database design with the knowledge that the operation network will be about as stable as the database.

This has not been attempted. It would be interesting and beneficial to investigate whether it is so.

D5.3 SPECIFICATION OF BUSINESS RULES

Business 'rules' are implicit in an entity relationship model of an organisation. These are relevant to an associated action model but do not need to be repeated in the action model. For example, if entity type A has a mandatory relationship with entity type B, then an occurrence of A must not be deleted without deleting the related occurrences in B; the description of this is explicit in the entity relationship model so any checks in the action model can be implicit. So issues that tend to be confusing for non-computer-oriented users, can be dealt with by defaults captured directly from the data model, thus easing communication.

Research is required into the implications of this. Some people think that it is possible to specify everything in an entity relationship model so no activity models are needed, especially detailed ones. I dispute this on an intuitive level, but cannot prove it. This would be a worthwhile piece of further research though.

CHAPTER E
A DIAGRAMMER FOR THE AUTOMATIC PRODUCTION OF ENTITY RELATIONSHIP DIAGRAMS

CHAPTER E A DIAGRAMMER FOR THE AUTOMATIC PRODUCTION OF ENTITY RELATIONSHIP DIAGRAMS

E1 INTRODUCTION TO THE DIAGRAMMER

One of the main benefits of Entity Analysis is that its resulting models have diagrammatic representations. This is useful because much more information can be meaningfully conveyed by diagrams than by the written word in the same space and is an important consideration because the diagrams are mainly used as communication tools, particularly for validation by users.

The use of diagrams introduces the problems of production and maintenance, on which a large amount of Analyst's time can be spent. The production of diagrams may introduce errors due to human fallibility; changing diagrams by hand also increases the chance that errors occur. In addition, redrawing introduces the problem of version control of the diagrams. Thus the manual production of diagrams can lead to reduction in the quality of information portrayed. A further problem which is encountered when diagrams are hand drawn is a reluctance on the part of a drawer to change their diagram, especially when complex; analysts have been known to argue against change rather than redraw their diagrams. This chapter describes an automated aid for diagram production on a micro-computer; the aid is called a 'Diagrammer'. It was produced in the initial part of this research project.

The Diagrammer is a suite of programs for the automatic production of diagrams which use entity relationship diagram diagrammatic conventions (see appendix X1). As such, the work presented in this chapter is in the context of producing entity relationship diagrams only. As will be discussed in E6, action models can also be produced using the Diagrammer.

The use of a diagrammer is analagous to the use of a simple Data Dictionary (ie. a Data Dictionary containing just definitions). A Data Dictionary is a central repository of definitions of data, activities etc. which enables this information to be interrogated [DDSWP,74]. A diagrammer has a central repository of the information to be drawn, eg. entity types and their relationships, actions and their dependencies, and produces diagrams for enquiry purposes.

The Diagrammer consists of two PASCAL programs, one to produce the diagrams and the other to view them. It was initially developed on an APPLE II micro-computer and there is an improved development on a SIRIUS micro-computer. An IBM PC version also exists using a BASIC view program. Appendix X6 contains a listing of the programs.

This chapter first considers the principles and constraints which have affected the design of the Diagrammer. Following this some of the programs' technical details are discussed.

E2 DIAGRAMMER PRINCIPLES AND CONSTRAINTS

There are various principles and constraints that have dictated the design of the Diagrammer.

E2.1 DIAGRAMMER & MICRO-COMPUTERS

A requirement of the Diagrammer was that it should be implemented on a micro-computer. The reasons for this are the ability to take a micro-computer into interview situations, the ease of using graphics on this medium, the ubiquitous nature of micro-computers and the portability potential of the program and any diagrams produced.

E2.2 GRAPHICS PACAKAGE CONSTRAINTS

Any implementation of a diagrammer is heavily constrained by the particular graphics package it interfaces with. However this does not affect the content of a diagram, just how the diagram is viewed. For example, even with the smallest character set available in the graphics package used for some of the figures in this chapter, truncation has been forced in the display of some entity type names rather than overwrite symbols; eg. TECHNICIAN in figure E2.2 appears as TECHNICI. Also, the characters stay the same size when the diagram is scaled up or down.

E2.3 HUMAN IMITATION

One of the aims of the Diagrammer is to imitate human diagram production methods in producing diagrams and this has been attempted where possible. However the limitations imposed by the use of micro-computers has constrained this aim considerably.

E2.4 DIAGRAM PRODUCTION METHODS.

There are two reasonable methods of automating the production and maintenance of diagrams: interactive and automatic.

E2.4.1 Interactive Diagram Production

There are a number of programs which enable the interactive 'building' of diagrams, for example see [TBT, 83], and there are also a number under development. The principle behind this method is to interactively 'draw' the diagrams by selecting desired symbols from a menu, eg. figure E2.1.

The main benefits of this method are ease of input and maintenance and a solution to the version control problem. The main deficiency of this method is that human errors can still be introduced, although the number of such errors will probably be decreased.

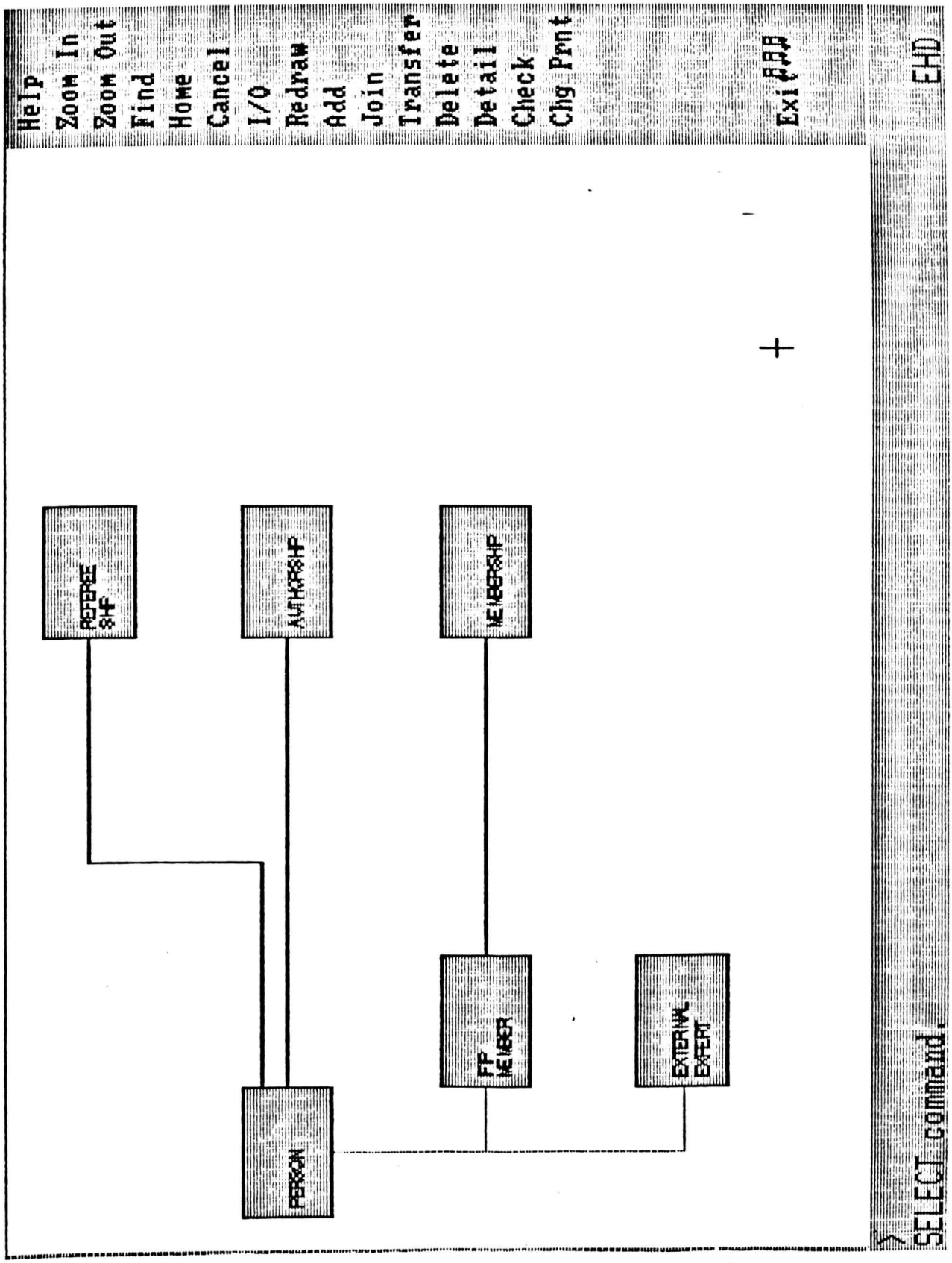


Figure E2.1 A Typical Interactive Building Screen

E2.4.2 Automatic Diagram Production

The second method of diagram production is to automatically produce a diagram from a description of a model; the Diagrammer is based on this method, but allows interactive building as well. The description input to the Diagrammer is a list of entity types, their relationships, and the optionality and cardinality of the relationships. The output from the Diagrammer is a diagram, eg., figure E2.2.

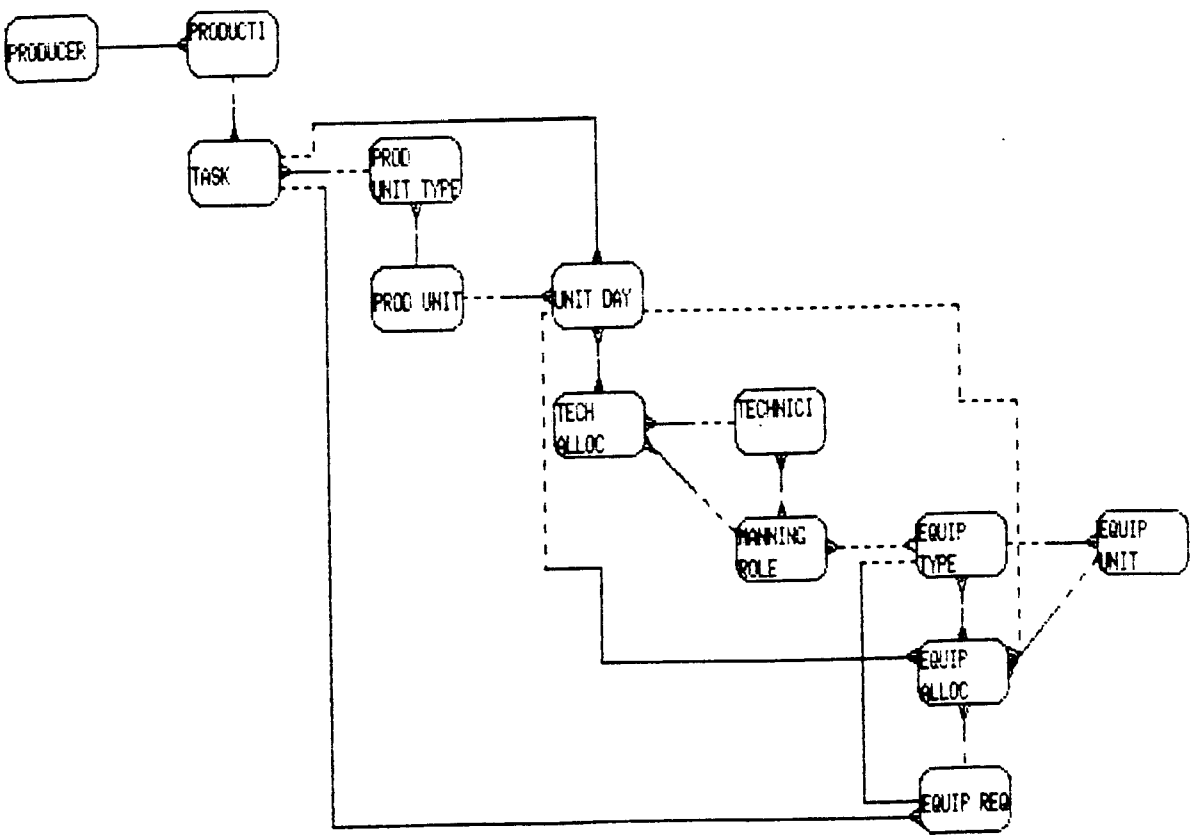


Figure E2.2 An Example of The Output of The Diagrammer

The main benefit of this method is the lack of human interaction, which results in many fewer production and maintenance errors. An automatically produced diagram is a true reflection of the originating input. For example, there is no possibility of forgetting a 'many' symbol, optionality circle, relationship, or entity type, all of which can easily be missed in a hand-drawn diagram; the only source of errors is in the input itself. Furthermore, if mistakes are subsequently detected (eg. by further analysis), it is much easier to automatically reproduce a diagram than it is to manually redraw parts of, or even all of a diagram. Diagram version control can also be easily achieved using automatic production.

Automatic and interactive diagram production methods are complementary; so models can be produced either automatically or interactively, and then interactively manipulated or automatically reformatted as desired.

E2.5 CRITERIA FOR DIAGRAM PRODUCTION

The main problem with the automatic production of a diagram lies in the nature of the models which are represented. One of the strengths of analysis models is that they represent a view of the 'real world', so the diagrams of the models must be able to easily convey any underlying meaning. Analysts achieve this by reflecting the semantics of a model in their diagrams, for

example, an Order Item entity type would be placed closer to its owner, Order, than to the resulting Delivery. The Diagrammer does not take this into account because it would require extremely complex semantical representations to be used and processed.

It is just as complex for a computer program to deal with aesthetics; this problem is even difficult for people, eg. what makes model A nicer to look at and/or able to communicate more than model B. Every Analyst has their own diagrammatic style: for example, Ellis [ELLIS, 82] insists a 1:Many relationship should be drawn with the many on the left wherever possible, eg., figure E2.3; others will put the entity type with the most relationships in the middle of a diagram, as in figure E2.4; and so on. The Diagrammer deals with this problem by giving a user a choice of a number of optional guidelines for the layout of a diagram. For example, figures E2.4, E2.5 and E2.6 all depict the same model, ie. they are isomorphic diagrams. The alternative layouts were produced as a result of different user options being chosen; see E3.2 for an explanation of the options provided by the Diagrammer.

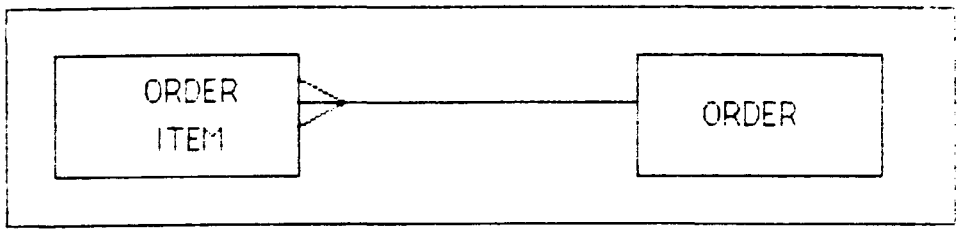


Figure E2.3 Ellis Style Relationship

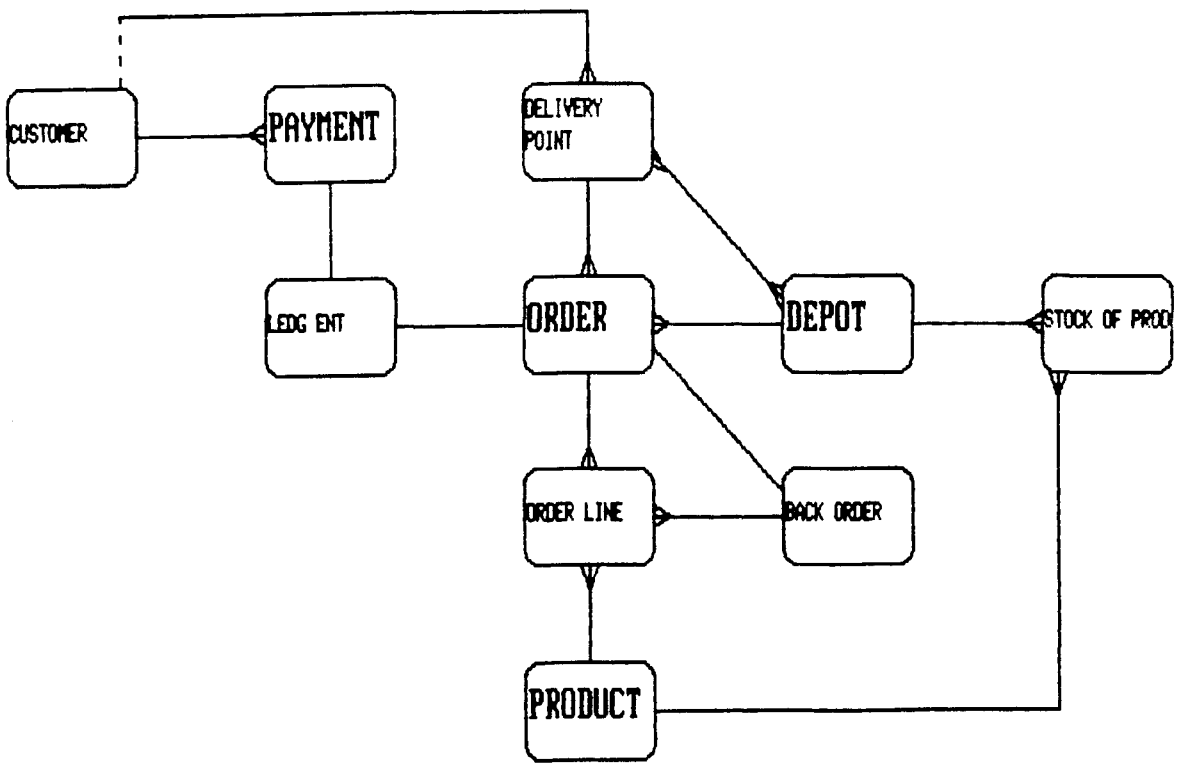


Figure E2.4 Example of Isomorphic Diagram
No User Option Chosen

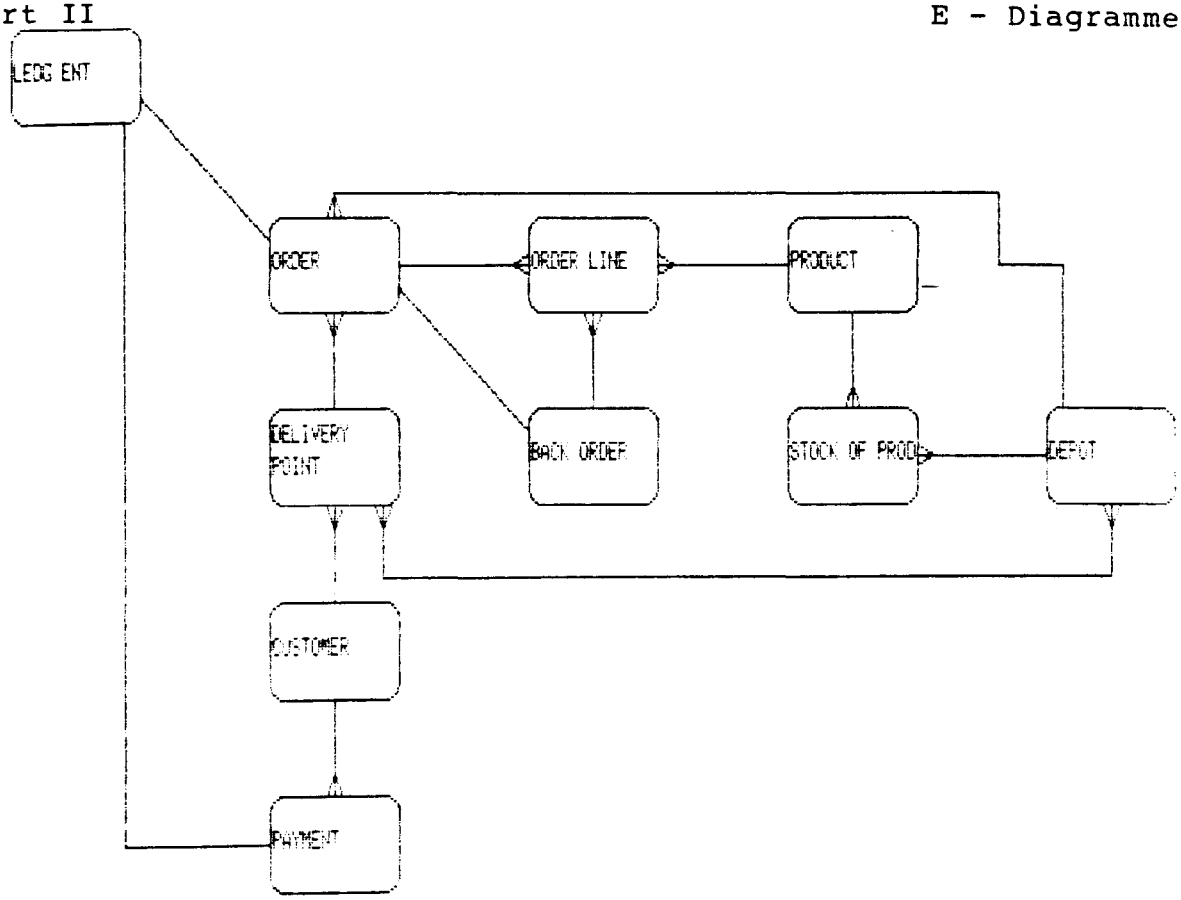


Figure E2.5 Example of Isomorphic Diagram
User Option - Relationship Priority/Maximum Entity Priority

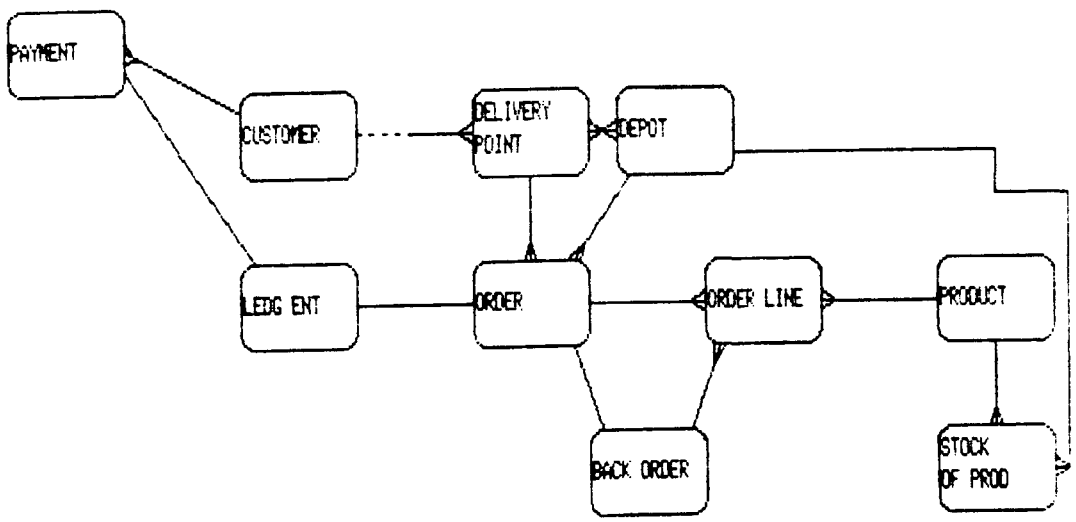


Figure E2.6 Example of Isomorphic Diagram
User Option - Entity Priority/Average Entity Proximity

The criteria that are thought to affect the communication potential of a diagram are the number of crossing lines and the length of relationships. The Diagrammer tries to reduce both of these factors. This is achieved by a 'semi-intelligent' relationship connection algorithm (see E3.3). Various graph theories could have been utilised to achieve a similar result as has recently been done by Tamassia et al [TBT, 83]. Initial research for the Diagrammer rejected the use of graph theory because it was felt that it would restrict the ability to produce isomorphic diagrams. Furthermore, as very few analysts utilise graph theory in drawing diagrams, it was felt that it should be possible to produce reasonable diagrams without considering graph theory.

E2.6 DIAGRAMMER INPUT

The Diagrammer is aimed at the utilisation of information contained in a Data Dictionary-like environment. A separate input interface is provided to enable this. In the prototype the input interface uses a file containing the necessary information. However the interface can be easily adapted to read the required information from any Data Dictionary.

E2.7 VIEWING LARGE DIAGRAMS

Facilities are required to view any diagram which does not easily fit on a screen. The size of diagram which can fit on a screen depends on the resolution properties of the screen and the graphics package used. The minimum set of viewing facilities are to scroll a diagram left, right, up and down, to zoom in to concentrate on a section of a diagram, or to zoom out to see more of a diagram, and the ability get hard-copy of the diagram. Optional facilities which may be useful are diagonal scrolling, to be able to scroll pages, or just small amounts, and rotation of the diagram to gain different perspectives.

A problem which has been recognised with viewing diagrams on paper is the difficulty of comprehending large amounts of information in a small space; this problem is intensified when viewing on a screen. For entity relationship diagrams this problem can be overcome by the use of "Entity Model Clustering" (see chapter C). Entity model clustering is very suitable for automation and solves the problem of viewing large entiy relationship models while easing the burden on the production of the diagrams through fewer entity types having to appear.

E3 TECHNICAL DETAILS

This part describes the more significant technical issues since it is inappropriate to dwell on the technical design details in any length.

E3.1 INTERNAL REPRESENTATION OF THE DIAGRAMS

As the Diagrammer is aimed at micro-computers, good utilisation of memory is of the utmost importance. To achieve this a diagram is held as a set of linked lists. This is also due to a need for flexibility of different size models. The alternative is to have a fixed size matrix; this matrix would be very sparse,leading to an enormous amount of wasted memory. Various other alterntives were ruled out due to speed considerations.

E3.2 LAYOUT OF ENTITY TYPES

Entity types must be placed so as to be close to related entity types while avoiding overcrowding. The following describes how this is achieved.

The entity types are first ordered according to some criteria, for example, the criteria could be ordering by the number of relationships an entity type has, or simply the order of input (perhaps enabling some user preferred criteria). The prototype has three options which affect the layout of a diagram. One

option affects the ordering; this is Relationship Priority.

For this option the entity types are ordered to be in ascending number of relationships (called Minimum), or descending number of relationships (called Maximum, eg., figure E2.5). These cause the entity type with the minimum or maximum number of relationships to be placed on the diagram first. The second option is Entity Priority (eg. figure E2.5 and E2.6). This affects the order in which entity types are taken from the ordered list. If Entity Priority is not chosen the entity types are placed in the order of the list and relationships are only connected to entity types already on the diagram. With Entity Priority the entity types are also taken in order, but all of an entity type's relationships are connected, even if this necessitates the placing of other entity types out of a chosen order. If this occurs all of the relationships of the entity types that are taken out of order are connected; this may require taking even more entity types out of the chosen order, and so on.

The third user option is Average Entity Proximity (eg. figure E2.6). On layout, each entity type is placed as close to an already placed related entity type as possible. However, if Average Entity Proximity is chosen, an entity type is placed so as to be as close as possible to a position which is the average of all the already placed related entity types'positions.

If no related entity type has yet appeared on the diagram, an entity type is placed as close to a mythical origin as possible. The constraints which affect the placing of entity types are that an entity type must not be placed so as to overlap another entity type or to be on top of a relationship. The search for a free position proceeds in ever increasing circles whose centre is either the related entity type, the average position, or the origin. The start point on the intial search circle is varied so as to avoid overcrowding in a particular area.

The introduction of lookahead and/or backtracking techniques has been considered to improve the layout of entity types. People tend to employ these techniques when drawing these diagrams. For example, first thought is first given to the effects of putting an entity type at a given place on the diagram; if it is felt to be satisfactory the entity type will be placed and built round; if the further placing of entity types results in the diagram becoming unsatisfactory, much rubbing out and redrawing will occur. Backtracking and lookahead have not been introduced at this stage because it would appear that the vast projected increases in space and time needed to cope with them would be unreasonable.

E3.3 RELATIONSHIP CONNECTION

Relationships are drawn so as to meet the aesthetic criteria detailed in E2.5, which are to reduce the length of the

relationships and to minimise the number of crossing lines. In addition entity type boxes must not be crossed. The connection of relatationships is achieved via a spatial search algorithm employing 'tree-pruning'. The algorithm is based on the generalisation that only two of an entity types four sides are useful when aiming towards another entity type, one vertical and one horizontal; for example, in figure E3.1, sides 1 and 2 of A are most suitable for starting relationships towards B.

Four relationship paths are attempted in all - one from the two most appropriate sides of each entity type, eg., in figure E3.1, one from 1 & one from 2 towards B and one from 3 & one from 4 towards A. The path chosen is the one which is most satisfactory. The main aim of the algorithm is to reduce the distance between the current end of a path and the target entity type.

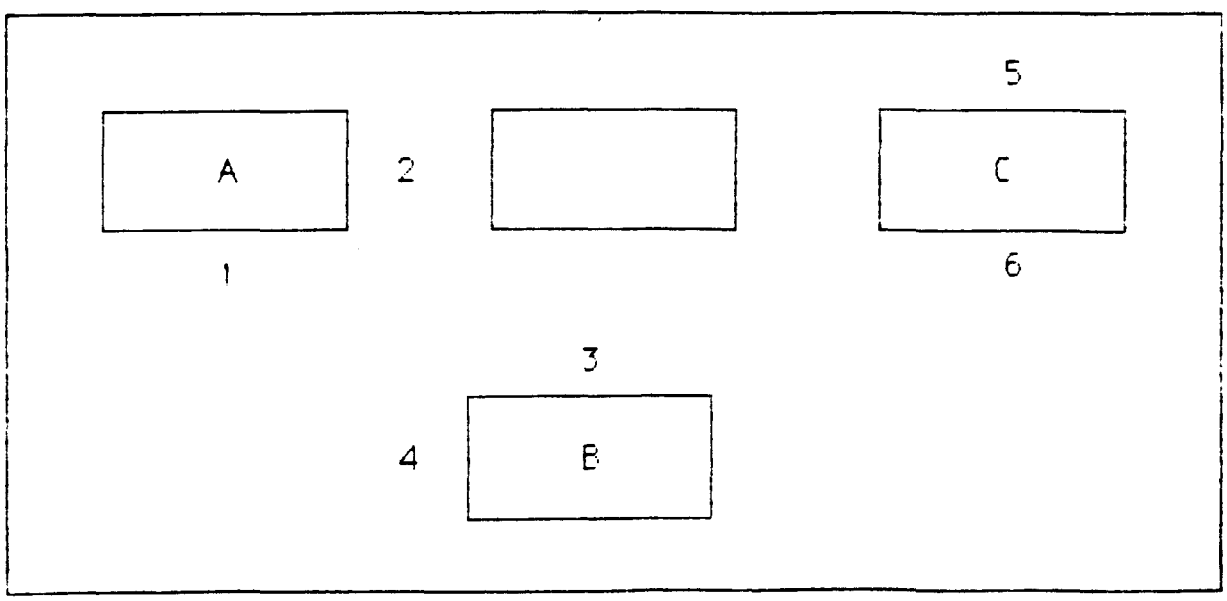


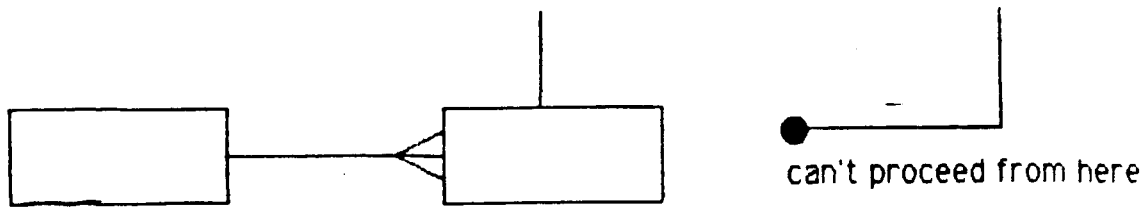
Figure E3.1 Useful Start Sides

First of all a starting place for a path is found on the side to be started from; the path is abandoned if there is no space for any more relationships on that side. Next a path is drawn in an orthogonal direction to the starting side; the point drawn to is the closest point to the target entity type that does not lead to the path intersecting any other entity types. This process is then repeated with the point just found being substituted for the starting point and the path heading in an orthogonal direction to the previous path. Eventually the path either becomes untenable (ie. cannot proceed without crossing an entity type box), or the target is reached. A 'score' is calculated while constructing the path to enable the best path to be found, this being the path with the minimum score. The score includes a value for every line in the path to penalise long, complex relationships and a value for every relationship crossed. The search tree is 'pruned' by abandoning the search for a path if its score ever exceeds the minimum score achieved in previous path searches for the relationship. If two paths have the same score, the chosen path is the one which was explored first.

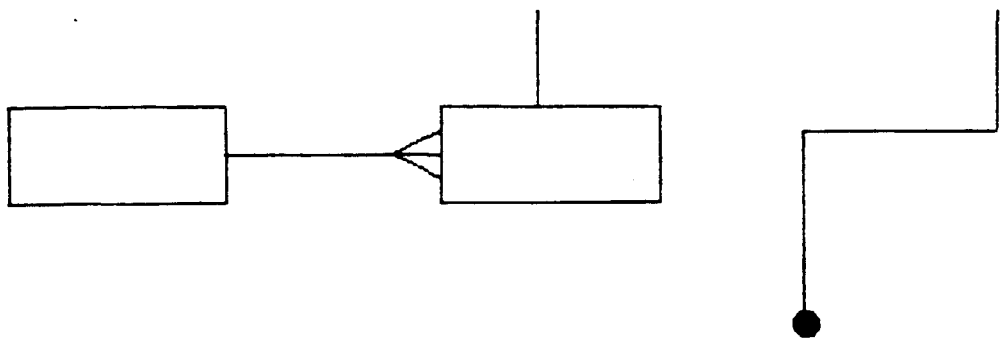
The algorithm considers many special cases. For example, a proposed path might occupy the same space as an already completed relationship. If this occurs, the proposed path is moved to one side and a penalty added to its score to discourage the selection of the path; the selection is discouraged because it can be difficult for a viewer to follow two paths which are close together. Another special case is where a point is

reached on a horizontal or vertical line to the target entity type, but with an intervening entity type, eg. figure E3.2(a). If the end of a path is in a direct line to the target any indirect path will increase the distance between them, violating the main aim of the algorithm. When this situation is recognised a violation is forced by drawing a line at right-angles to the desired direction, as in figure E3.2 (b,c).

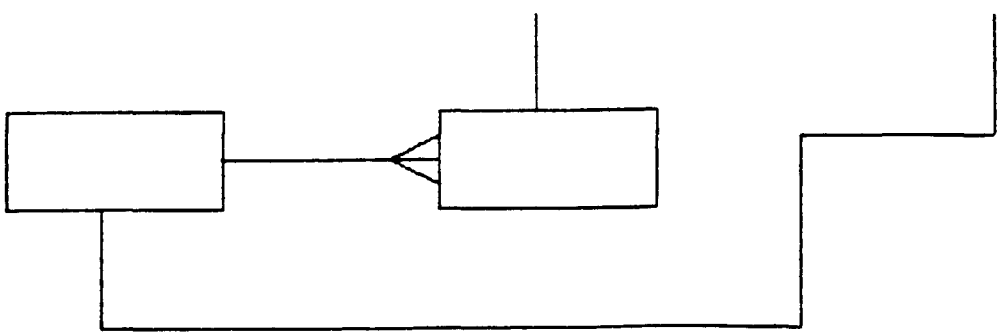
This also applies to the common case of entity types on vertical or horizontal lines with each other but with intervening entity types, eg., figure E3.3. However an extra path is attempted for this case as paths from either parallel entity type box side can be equally usable, eg., in figure E3.1 a path from sides 5 and 6 of C are both of value when heading towards A.



a) Path can't proceed without increasing distance



b) Diagrammer draws orthogonal line



c) Final Result

Figure E3.2 Intervening Entity Type Between End of Path & Target

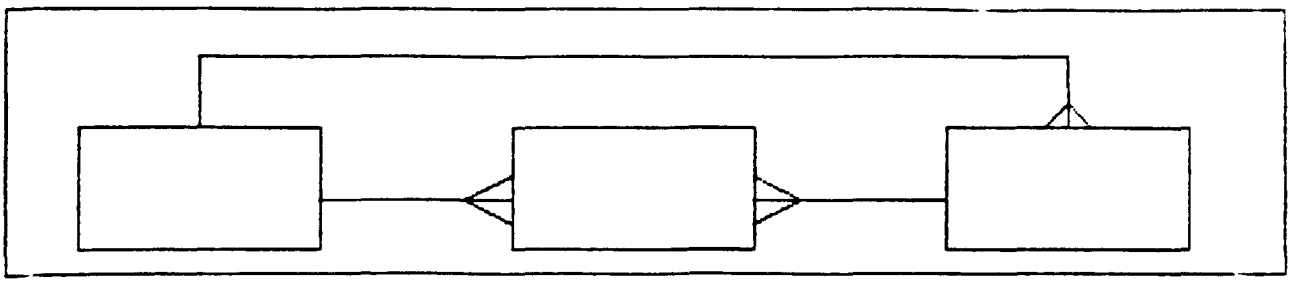


Figure E3.3 Intervening Entity Type Between Two Entity Types

A further special case is where a single diagonal line would suffice between two entity types. The use of such a line is very much a matter of personal taste. The Diagrammer searches for such lines before attempting any of the other paths. A diagonal line is drawn if it does not intersect any other symbols, including relationships. Relationships are included in this case due to a widely stated preference that relationships should cross at right angles. Unfortunately it does not stop further relationships crossing an already placed diagonal line, but this is discouraged by having a higher penalty for crossing a diagonal line than for a horizontal or vertical line.

E3.4 VIEWING OF DIAGRAMS

The viewing of the diagrams is achieved using a separate program to the construction program. The main justifications for this are: to enable diagrams to be constructed just once, but viewed as often as desired, to enable a common viewing program for different constructing programs, and to enable portability across micro-computers and graphics package as all the machine dependent and graphics oriented instructions only deal with viewing. The separation has the added advantage that a single construction program can deal with many different types of models as any changes for different symbols will occur in the viewing sections.

The diagrams are displayed with an optional menu of viewing commands, see figure E3.4. The Diagrammer's viewing facilities are: to zoom in to see less of a diagram or zoom out; scroll the diagram left, right, up or down, slowly or quickly (dot or half-screen at a time respectively) and diagonally slowly; to enable optionality to be shown on relationship lines or not (eg. for a strategy study); and to enable the digram to be printed out.

E3.5 MANIPULATING DIAGRAMS

An extra module was added to enable the manipulation of diagrams which were produced. This module was added quickly, but worked

fairly well. The basic facilities added were: to add an entity type, move its position on a diagram, delete an entity type, add, move or remove a relationship, and change the optionality and cardinality of a relationship.

This facility can also be used to interactively produce a diagram.

E4 USE OF DIAGRAMMER

The Diagrammer created a lot of interest when it was produced. It was going to be produced commercially by CACI Inc International with a leading building society, but CACI changed its management at that time and the project was dropped. Shortly after I joined JMA which had diagramming tools of its own (though no automatic production at that time - it now does have as part of the Information Engineering Facility [TI, 86]). Due to this the interest in the Diagrammer dropped from the project and the areas discussed in previous chapters were concentrated on. I have been told that CACI have since produced the Diagrammer as a commercial product in conjunction with the University of Delft.

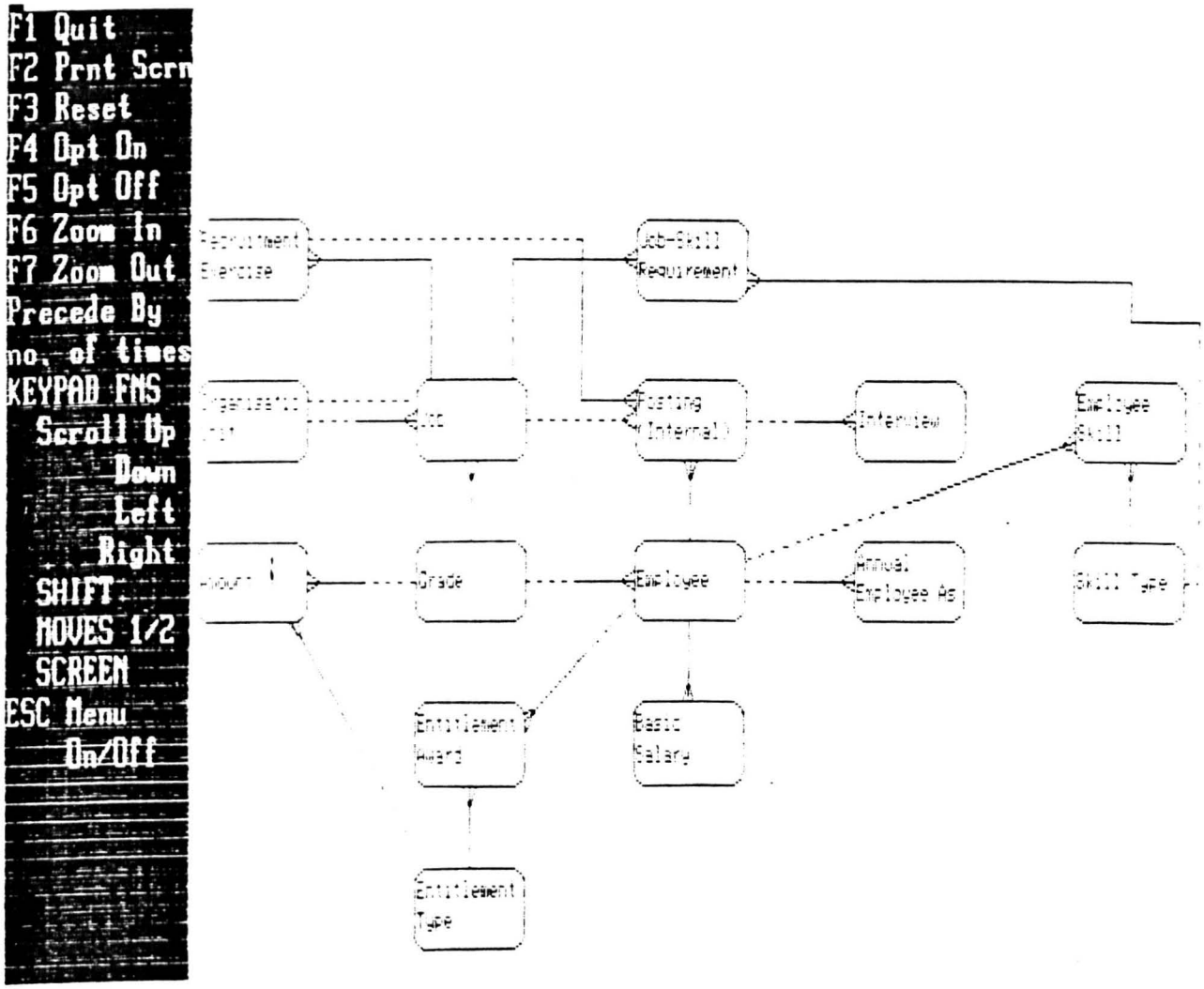


Figure E3.4 Example of Facility Menu

A THEORETICAL AND PRACTICAL INVESTIGATION OF TOOLS AND TECHNIQUES FOR THE STRUCTURING OF DATA AND FOR MODELLING ITS BEHAVIOUR

Volume III of III

by PAUL BARRIE FELDMAN B.Sc(Hons)

a PhD thesis

UNIVERSITY OF WARWICK
COVENTRY, ENGLAND

SCHOOL OF INDUSTRIAL AND BUSINESS STUDIES

September 1986

PAGE NUMBERING AS ORIGINAL

C O N T E N T S

LIST OF ILLUSTRATIONS

PART III

CONCLUSIONS

CHAPTER F
RESEARCH METHOD

CHAPTER F RESEARCH METHOD

F1 INTRODUCTION

This section describes the history of the research, the way I went about it and the problems faced. The section is structured as:

* Background leading up to the research
* Carrying out the Research.

There was no particular research methodology employed due to the nature of the research. Basically the work was done on a 'trial-and-error' basis with a 'longitudinal study' for entity model clustering.

Being a collaborative effort, the main driving force of the research was to provide practical techniques that worked and had benefit. This may have detracted from the academic neatness of the research, but this was felt to be worthwhile in the context.

F2 BACKGROUND TO RESEARCH

CACI Inc International (henceforth CACI) developed their D2S2 Analysis Methodology over a number of years [ROEV,81], [MACP,82]. The majority of this development was undertaken in a practical environment. Although there was research input [DAVE,80], [SHAVE,81], it tended to be a documentation of existing situations rather than providing new methods or insight. The techniques were mainly developed for database design information gathering and occasionally this produced unsatisfactory results. Due to the aim of design information gathering, the information required from the analysis process is sometimes obscure with no obvious reasons for its collection. A classic example of this is the collection of relationship percentages; nowadays CACI personnel are unsure what they are for. Due to a turnover in staff the original D2S2 development team left the company, taking with them their expertise of the methodology details. These plus other reasons prompted CACI to sponsor research into D2S2. The initial proposal is attached as appendix X7. The research proposal resulted in this SERC CASE award.

My previous experience was a larely theoretical degree in computational science and various programming jobs. During this work experience none of the projects I was involved in were successful, this being largely due to lack of analysis. This led to a disquiet with the way development took place and

resulted in me applying for the opportunity to take up the CASE award. So the research started from the premise that things were not right and that there was a lot of room for improvement.

F3 RESEARCH HISTORY

The research started in September 1982. I spent the first month at CACI Inc. International, getting to know the subject and attending a course in 'data analysis'. One aim of this month was to provide a list of possible research areas by discussing data analysis with the consultants and identifying any problems they thought existed.

This was achieved by interviewing some consultants and separating out genuine concerns from personal 'bug-bears'. In fact the most common reponses to the question "What do you think needs researching?" were "I don't know" and "nothing". However one of the main concerns at the time was the automation of analysis, as this was about the start time for CACI's 'The System Factory' concept - more of which later.

At the end of the month a list of possible research areas was drawn up and included:

i) Data Analysis for distributed systems
ii) Determination of data model equivalence
iii) Automatic generation of 'TNF relations'
iv) Data Anlaysis for the design of Organisation structure
v) Automatic diagramming of entity relationship models.

At the end of the month I started at Thames Polytechnic. Here

the area of modelling data behaviour (vi) was added to the list, largely because it was felt that D2S2 placed such a large emphasis on data modelling that the activity modelling aspects were downgraded too far. The aim of this area was to investigate how activity modelling could be on a par with data modelling. The result of this investigation was the development of the action modelling technique discussed in chapter B and investigation of the implications of data behaviour modelling (vii) was added as a result.

The investigation of these areas then proceeded. First a literature seach in each area and in related aspects was carried out (and has continued to be for the areas chosen to be more deeply researched) and basic investigation of solutions was attempted. In area (v) a set of programs were produced to 'draw' entity relationship diagrams (the results of which can be seen in chapter E).

About March 1983 Whitbread & Co. Plc approached Thames Polytechnic for a consultant to help them consolidate a set of data modelling studies they had done. This resulted in the entity model clustering technique (viii) described in chapter C (the actual Whitbread work is also described in this chapter C6.1).

For this area and all the others a set of papers was produced in March 1983 describing the initial literature searches and

investigations [FELD,83]. There was not enough time to investigate all of these areas in depth, though it was felt that they all had research potential. The areas that were investigated further, action modelling, entity model clustering, and the diagrammer, have all been described previously.

The remaining areas all formed interesting research areas with a lot of potential. The automatic generation of relations was rejected due to the large amount of other research that had been done and is being done to achieve it.

The initial investigation into the use of data analysis for the design of organisational structure showed there was a lot of potential. The premise behind this area was that an organisation's structure should reflect the information in use, this being backed by 'expert' literature in organisational design, though not to as high a degree as was proposed. However the research team did not possess the experience or resources to be able to investigate the area fully, so it was grudgingly rejected.

The physical implications of action modelling was relient on investigating action modelling first, so could not be investigated in isolation. When action modelling was chosen for research it was decided that there were insufficient resources to do justice to this 'physical' area as well. This area, called 'operationbases', was the investigation of a 'database' for

actions with an attempt to institute the same regimen for actions as is now achieved for data. The results are similar to program libraries and abstract data types, but an operationbase is to these, as a database is to a filing system.

The investigation of data model equivalence would have been both fruitful and useful, the track which was initially investigated has since been proved to be largely correct. However papers were subsequently published in this area reducing a large part of its originality.

The final area considered was analysis for distributed data sharing systems. This was an attempt to investigate the requirements that a distributed system would place on the analysis process. It would also have been fruitful and useful. A decision had to be taken whether to reject action modelling, entity model clustering, or distribution. Distribution was rejected mainly because a full investigation would be dependent on techniques such as action modelling and entity model clustering.

The research on the chosen areas proceeded over the next few years. During which time the research base moved to the University of Warwick, due to a move by my supervisor, Guy Fitzgerald.

One impact on this research was a consideration of the

automation of analysis. This was largely due to CACI Inc. International's efforts to produce a development workbench called 'The System Factory' [CACI, 83]. The intention of this workbench was to capture analysis data, convert it to design data, allow this to be manipulated and finally to generate a working system. I was closely associated with this development, though the only part of my research which would have had an immediate impact was the diagrammer. The diagrammer was in fact demonstrated at System Factory demonstrations as 'the future'. (The System Factory was totally menu and text-based, there were no graphics.) However due to my close involvement in this area automation was considered for the other techniques.

Unfortunately in the spring/summer of 1984 the System Factory development was closed down and the CACI division I was associated with disbanded. I had effectively lost my sponsoring associates. This had two effects. One was that I decided to become a more-or-less full-time consultant myself (initially at CACI) and to continue this research on a part-time basis, adding an extra year to the timescales. The second effect was a lessening of the emphasis on automation as there was no longer any opportunity to attempt it. The diagrammer actually continued to be developed at CACI, but very much in the background. Towards the end of 1984 I received a job offer from James Martin Associates (JMA) which I accepted. I heard no more about the diagrammer until the summer of 1985 when I met a CACI Consultant who informed me it was being developed in Holland in

conjunction with the University of Delft. The diagrammer had fallen away as a research issue for me because JMA were (and still are) developing such tools in conjunction with Texas Instruments [TI, 86].

During 1985 and 1986 the research into entity model clustering and action modelling was consolidated and used in anger where possible (see B6 and C6 for details). The ideas were published (see appendix X5) and discussed with other consultants whenever possible. During this time I persuaded JMA of the usefulness of the techniques and incorporated them into the Information Engineering methodology, mainly in its Business Area Analysis stage.

The 'parallelism' theory came about a result of the other developments. We realised that both main areas had in common an emphasis on the closeness of data and activity. When investigated this resulted in the discussion contained in chapter D.

This thesis details the latest state of the research.

CHAPTER G
RESEARCH PROBLEMS

"In my branch of learning, experiment is impossible. Advance is made through thought. Experiment is often used as a substitute for thought."

Patrick Steptoe - The Guardian Mon. April 22 1985

CHAPTER G RESEARCH PROBLEMS

This chapter discusses particular problems faced in carrying out this research.

By far the biggest problem was trying out the research in a commercial environment. As the research was intended to be of practical use this had to be done, but nobody wanted to be a guinea pig. There was less problem with entity model clustering because it had been developed in a commercial institution; this did not stop people doubting its usefulness until they had tried it though.

Action modelling was extremely difficult to use, as discussed in B6. Few projects dealt in this area and those that did, did not want to try it. The sticking point was that the benefits of action modelling are much harder to see than the benefits of entity model clustering. Eventually action modelling was used, but it was a battle. I have reached the conclusion that system developers are essentially very conservative and dislike or distrust any idea that seems like an intrusion or seems novel.

One of the reasons that the research had to be used in a practical environment was a problem of validation. How do you 'prove' that a technique is right? There are no criteria for evaluating techniques. As with Steptoe, experiment is often

impossible in this branch of learning as well and has to be by thought. Mathematics can be used to an extent (for example, see appendix X3), but in information systems development it appears that almost anything can be justified by choosing a flexible enough set of axioms. So how else do you prove a technique? Use in a practical situation is one, very good method. If a technique provides the expected answer then it is empirically proven. It could be said that any technique that gives some benefit and does not cause more problems, works and is right. However it is not sufficient just to use a technique, its affect on the environment and its continuing impression must be taken into account. This was done at Whitbread in what amounts to a longitudinal study over three years. The results were very encouraging because entity model clustering has now become a part of the Whitbread development culture.

Another way to validate techniques is peer judgement. This can take two forms: academic acceptance and practitioner acceptance. The former was achieved in this research by publishing all the results in refereed journals or conferences (see appendix X5). Practitioner acceptance was discussed previously. Basically it was achieved by persuasion, example and counter-example.

This discussion raises a further question of, is information systems analysis research a science or a craft? The necessary emphasis on practical use and lack of theoretical basis leads me

to consider it a craft. Therefore I believe the validation I have achieved is all that can be done.

CHAPTER H
CONCLUSIONS OF RESEARCH

CHAPTER H CONCLUSIONS OF RESEARCH

This document has discussed the content and progress of a collaborative research project in the area of information systems development. The main objective of the project was to investigate various areas based around the technique of entity relationship modelling to see if they could be improved, given a theoretical background, and/or automated.

To this end the investigation focussed on three products: entity model clustering, action modelling, and an automated tool (the diagrammer) all of which are discussed in the preceeding chapters. The main products, entity model clustering and action modelling, attacked the areas of improving the use of entity relationship modelling in complex and diverse situations and improved activity modelling by giving it a rigorous technique which built on the success of entity relationship modelling. These two areas were brought together by the development of theorems concerning data/activity symmetry. The theorems are:

> "Every data object has an equivalent activity object and vice-versa"

and

> "Every data modelling structuring concept has an equivalent activity modelling construct and vice-versa".

The theorems could only be developed after the main research had been done because both the techniques provided a 'missing link' in the symmetry. Entity model clustering developed subject areas, an equivalent to the activity concept of function. Action modelling developed actions and their dependencies which are equivalent to entity types and their relationships. There were no similar concepts beforehand to fill in the symmetry holes.

Entity model clustering and action modelling have both been used in practice, entity model clustering more so than action modelling. Persuading practioners to use the techniques was one of the main problems of the research; entity model clustering has more obvious immediate benefits so was an easier subject for persuasion.

Both techniques have been incorporated into James Martin Associate's Information Engineering Methodology to an extent. As entity model clustering has more 'charisma' than action modelling it was easier to introduce to the methodology and has been incorporated in its entirety and unchanged. Action modelling was introduced on the back of a less detailed technique, process dependency analysis, and had to be adapted accordingly (eg. terminology had to be changed and it was incorporated as detailed analysis below elementary process level). Some of the more academic parts of action modelling, such as forcing a matching of data and activity hierarchy, are

still to be incorporated.

Both main areas have been well-researched. There is little left to work on except the implications of the techniques when particular modelling issues are considered. Both other areas, the symmetry and the diagrammer, would warrent further research as the subject of their own research project, especially the symmetry aspect. Both minor areas are really offshoots of the research.

One aspect which has hardly been touched at all is the theoretical basis for entity relationship modelling. This again could form a research project on its own, but with few immediate practical benefits. I expect the likely direction of this research would be the application of set theory concepts, which are highly similar in nature.

As mentioned earlier this project was collaborative research, so its main success criteria was the production of useful, worthwhile techniques and the furthering of the understanding of entity relationship modelling. I believe these criteria have been met as all the research areas have produced useable results, all the results have been published and all have furthered the general understanding of entity relationship modelling.

CHAPTER I
REFERENCES

CHAPTER I REFERENCES

The abbreviations used in the following list are:

Proc = Proceedings of

IEEE = Institute of Electrical and Electronic Engineers

BCS = British Computer Society

ACM = Association for Computer Machinery

CACM = Communications of the ACM

VLDB = International Conference on Very Large Data Bases

Vol = Volume

No = Number

CUP = Cambridge University Press

SIGART = Special Interest Group on Artificial Intelligence

SIGMOD = Special Interest Group on Management of Data

SIGSOFT = Special Interest Group on Software Engineering

[ABCDLVZ,83] ATZENI, P, BATINI, C, CARBONI, E, de ANTONELLIS, V, LENZERINI, M, VILLANDELLI, F and ZENTA, B, INCOD-DTE: A System for Interactive Conceptual Design of Data, Transactions and Events, in [CERI,83]

[ABLV,83] ATZENI, P, BATINI, C, LENZERINI, M and VILLANELLI, F, INCOD: A System for Conceptual Design of Data and Transactions, in [CHEN,83]

[ABRI,74] ABRIAL, J R, Data Semantics, in [KLKO,74]

[ACJL,81] ABBATT, J, CAMPBELL, C, JONES, A H and LAND, F F, New Approaches to Systems Analysis and Design, in Proc BCS 81, British Computer Society, 1981

[AIKI,83] AIKINS, J S, Prototypical Knowledge for Expert Systems, Artificial Intelligence, Vol 20,1983

[ALVEY,85] ALVEY, Programmers Annual Report, DTI, 1985

[ASMO,82] ASCHIM, F and MOSTUE, B, IFIP WG8.1 Case Solved Using SYSDOC & SYSTEMATOR, in [OLLE,82]

[ATCA,83] ATZENI, P and CARBONI, E, INCOD Revisited After the Implementation of a Prototype, in [DJNY,83]

[AVFI,86] AVISON, DE and FITZGERALD, G, An overview of Systems Development Methodologies, Blackwell Scientific, Oxford, 1986

[BACH,69] BACHMAN, C W, Data Structure Diagrams, Data Base, Vol 1, No 2, 1969

[BADA,77] BACHMAN, C W and DAYA, M, The Role Concept in Data Models, Proc 3rd VLDB, 1977

[BADW,82] BOLOUR, A, ANDERSEN, T L, DEKEYSER, L J & WONG, H K T, The Role of Time In Information Processing: a survey, ACM SIGART Newsletter 80, 1982

[BAGO,79] BALZER, R and GOLDMAN, N M, Principles of Good Software Specification and Their Implications for Specification Languages, in Specification of Reliable Software, IEEE Computer Society, 1979

[BAKER,82] BAKER, G, (ed), Data Analysis Update, British Computer Society Database Specialist Group, 1982

[BAKER,83] BAKER, G, (ed), Data Dictionary Update, British Computer Society Database Specialist Group, 1983

[BALZ, 80] BALZER, R, An Implementation Methodology for Semantic Data Base Models, in [CHEN, 80]

[BALZ,81] BALZER, R, Dynamic System Specification, in [BRZI,81]

[BBW,77] BALLY, L, BRITTEN, J N G and WAGNER, K, A Prototype Approach to Information System Design, Information and Management, Vol 1, 1977

[BC,85] BUTLER COX, The Effective Use of System Building Tools, Report Series No 47, Butler Cox Foundation, 1985

[BFN,85] BATINI, C, FURLANI, L and NORDELLI, E, What Is a Good Diagram? A Pragmatic Approach, in [IEEE,85]

[BIBR,84] BINGHAM, J E and BREEDON, R W, Bridge That Gap, Computer Bulletin, II/40, 1984

[BODE,83] BOLOUR, A and DEKEYSER, L J, Abstractions in Temporal Information, Information Systems, Vol 8, No 1, 1983

[BOJA,66] BOHM, C and JACOPINI, G, Flow diagrams, Turing Machines and Languages with only two Formation Rules, CACM, Vol 9, No 5, 1966

[BOPI,79] BODART, F and PIGNEUR, Y, A Model and a Lanaguage for Functional Specifications and Evaluation of Information System Dynamics, in [SCHN,79]

[BORK,80] BORKIN, S A, Data Models: a semantic approach for database systems, MIT Press, 1980

[BRIT,80] BRITTEN, J N G, Design for a Changing Environment, The Computer Journal, Vol 23, No 1, 1980

[BROD,80] BRODIE, M, Data Quality in Information Systems, Information & Management, Vol 13, 1980

[BROD,83] BRODIE, M L, Association: a database abstraction for semantic modelling, in [CHEN,83]

[BRSI,82] BRODIE, M L and SILVA, E, Active and Passive Component Modelling, in [OLLE,82]

[BRZI,81] BRODIE, M and ZILLES, S, (ed), Proceedings of Workshop on Data Abstraction, Databases and Conceptual Modelling, ACM SIGMOD Record, Vol 11, No 2, 1981

[BUBE,77a] BUBENKO, J A, The Temporal Dimension in Information Modelling, in [NIJS,77]

[BUBE,77b] BUBENKO, J A, Validity and Verification Aspects of Information Modelling, Proc 3rd VLDB, 1977

*[BUBE,80] BUBENKO, J A, Information Modelling in the Context of System Development, in Information Processing 80, (ed) S Lavington, North Holland, 1980

[BURC,85] BURCHETT, R, Data Analysis and the LBMS Structured Development Method (LSDM), Database Bulletin, Jan, 1985

[CACI,83] CACI, The CACI System Factory Product Summary, CACI Inc. International, 1983

[CCA,78] CENTRAL COMPUTER AGENCY, Computer System Development: 2 Project Management, CCA Management Note, No 4, 1976

[CEBR,84] CENDROWSKA, J & BRAMER, M, Inside an Expert System: a rational reconstruction of the MYCIN consultation system, in Artificial Intelligence: tools, techniques and applications, (ed) T O'Shea and M Eisenstadt, Harper and Row, 1984

[CERI,83] CERI, S, (ed), Methodology and Tools for DataBase Design, North Holland, 1983

[CHBO,74] CHAMBERLIN, D D, and BOYCE, R F, SEQUEL: A Structured English Query Language, Proc ACM SIGMOD, 1974

[CHEC,81] CHECKLAND, P, Systems Thinking, Systems Practice, Wiley, 1981

[CHEN,76] CHEN, P P, The Entity Relationship Model: towards a unified view of data, ACM Transactions on Database Systems, Vol 1, 1976

[CHEN,80] CHEN, P P, (ed), Entity Relationship Approach to Systems Analysis and Design, North Holland, 1980

[CHEN,83] CHEN, P P, (ed), Entity Relationship Approach to Information Modelling and Analysis, North Holland,1983

[CHLO,80] CHAN, E P F and LOCHOVSKY, F H, A Graphical Data Base Design Aid Using the Entity Relationship Model, in [CHEN,80]

[CLAN,83] CLANCY, W J, The Epistemology of a Rule-Based Expert System, Artificial Intelligence, Vol 20, 1983

[CLME,81] CLOCKSIN, W, and MELLISH, C, Programming in PROLOG, Springer-Verlag, 1981

[CODA,71] CODASYL, CODASYL Data Base Task Group Report, Conference on Data Systems Languages, ACM, 1971

[CODD,70] CODD, E F, A Relational Model of Data for Large Shared Data Banks, CACM, Vol 13, No 6, 1970

[DAVIS,80a] DAVIS, R, Meta Rules: reasoning about control, Artificial Intelligence, Vol 15, 1980

[DAVIS,80b] DAVIS, R, Content Reference: reasoning about rules, Artificial Intelligence, Vol 15, 1980

[DAVE,80] DAVENPORT, R A, The Application of Data Analysis - experience with the entity relationship approach, in [CHEN,80]

[DDSWP,74] BRITISH COMPUTER SOCIETY DATA DICTIONARY SYSTEMS WORKING PARTY, The British Computer Society data dictionary systems working party report, ACM SIGMOD Record, Vol 9, 1974

[DDSWP,82] BRITISH COMPUTER SOCIETY DATA DICTIONARY SYSTEMS WORKING PARTY, Journal of Development, 1982

[DEMA,83] DEARNLEY, P A and MAYHEW, P J, In Favour of System Prototypes and Their Integration Into the Systems Development Cycle, The Computer Journal, Vol 26, No 1, 1983

[deMA,78] DeMARCO, T, Structured Analysis and System Specification, Yourden, 1978

[DIJK,72] DIJKSTRA, E W, Notes on Structured Programming, in Structured Programming, (ed) O J Dahl, E W Dijkstra and C A R Hoare, Academic Press, New York, 1972

[DJNY,83] DAVIS, C, JAJODIA, S, NG, P A, RAYMOND, T Yeh, (eds), Entity Relationship Approach to Software Engineering, North Holland, 1983

[ELLIS,82] ELLIS, H, A Refined Model for the Definition of Systems Requirements, Database Journal, Vol 12, No 8, 1982

[ELLIS,85] ELLIS, H, Twenty Years of Data Analysis, in [HOLL,85]

[ESA,82] EUROPEAN SPACE AGENCY, ESA Software Engineering Standards, ESA BSSC (84) 1, 1984

[FEFI,85a] FELDMAN, P and FITZGERALD, G, Action Modelling: a symmetry of data and behaviour modelling, in Proceedings of the Fourth National Conference on Databases (BNCOD4), (ed) A F Grundy, CUP, 1985

[FEFI,85b] FELDMAN, P and FITZGERALD, G, Representing Rules Through Modelling Entity Behaviour, in [IEEE,85]

[FELD,82] FELDMAN, P, Initial Writings in Data Analysis, Thames Polytechnic Internal Publication, 1982

[FELD,83] FELDMAN, P, A Diagrammer for the Automatic Production of Entity Type Models, in Proc 3rd British National Conference on Data Bases (BNCOD3), (ed) J Longstaff, CUP, 1983

[FEMI,85] FELDMAN, P and MILLER, D, Entity Model Clustering, Database Bulletin, Jan, 1985

[FEMI,86] FELDMAN, P and MILLER, D, Entity model clustering: structuring a data model by abstraction, The Computer Journal, Vol 29, No 3, 1986.

[FERG,85] FERG, S, Modelling the Time Dimension in an Entity-Relationship Diagram, in [IEEE,85]

[FLAV,81] FLAVIN, M, Fundamental Concepts of Information Modelling, Yourden Press, 1981

[FLYNN,84] FLYNN, D J, An Analysis of Certain Data Models with Respect to their Handling of Selected Integrity Constraints, in Proc 3rd British National Conference on Data Bases (BNCOD 3), (ed) J Longstaff, CUP, 1984

[FMM,86] FELDMAN, P, MACDONALD, I and MABEY, C, Parallel Modelling of Data and Activity, to be published, 1986

[FSW,85] FITZGERALD, G, STOKES, N and WOOD, J R G, Feature Analysis of Contemporary Information Systems Methodologies, The Computer Journal, Vol 28, No 3, 1985

[GASA,79] GANE, C and SARSONS, T, Structural Systems Analysis: tools and techniques, Prentice Hall Inc, New York, 1979

[GILB,85] GILBERG, R F, A Schema Methodology for Large Entity-Relationship Diagrams, in [IEEE,85]

[GIMA,85] GIBSON, W E and MACDONALD, I G, The Corporate Database - automating the development process, State of the Art Report, Pergamon Infotech, Maidenhead, England, 1985

[GKB,82] GUSTAFSSON, M R, KARLSSON, T, and BUBENKO, J A, A Declarative Approach to Conceptual Information Modelling, in [OLLE,82]

[GOWI,80] GOLDMAN, N M and WILE, D S, A Relational Data Base Foundation for Process Specification, in [CHEN,80]

[GRAY,81] GRAY, J, The Transaction Concept: virtues and limitations, Proc 7th VLDB, 1981

[GRAY,84] GRAY, E M, An Empirical Study of the Evaluation of Some Information Systems Development Methods, University of Strathclyde, 1984

[GRIN,66] GRINDLEY, C, SYSTEMATICS: a non-programming language for designing and specifying commercial systems for computers, The Computer Journal, Vol 9, No 3, 1966

[GRIN,75] GRINDLEY, C, SYSTEMATICS: a new approach to systems analysis, McGraw Hill, 1975

[GRIN,79] GRINDLEY, C, The Role of the Trigger in Systematics, in [SCHN, 79]

[HALL,82] HALL, J, The LBMS System Development Method, in [BAKER,82]

[HAMC,78] HAMMER, M and McLEOD, D, The Semantic Data Model: a modelling mechanism for database applications, Proc ACM SIGMOD International Conference on the Management of Data, Austin, Texas, USA, 1978

[HAZE,75] HAMILTON, M and ZELDIN, S, Higher Order Software - a methodology for defining software, IEEE Transactions On Software Engineering, Vol SE-2, No 1, 1975

[HAZE,79a] HAMILTON, M & ZELDIN, S, Properties of User Requirements, in [SCHN,79]

[HAZE,79b] HAMILTON, M and ZELDIN, S, The Relationship Between Design and Verification, Journal of Systems and Software, 1979

[HEMC,83] HEITMEYER, C and McLEAN, T, Abstract Requirements Specification: a new approach and its application, IEEE Transactions on Software Engineering, SE-9, No 5, 1983

[HIRS,83] HIRSCHEIM, R A, Assessing Participative Systems Design: some conclusions from an exploratory study, Information & Management, No 6, 1983

[HOLL,85] HOLLOWAY, S, (ed), Data Analysis in Practice, British Computer Society Database Specialist Group, 1985

[IBM,75] IBM, Information Management System/Virtual Storage (IMS/VS) publications: General Information Manual, IBM, GH20-1260-3, 1975

[IEEE,85] IEEE, Proceedings of the 4th International Conference on Entity Relationship Approach, Chicago, IEEE Computer Press, 1985

[IIKE,83] IIVARI, J and KEROLA, P, A Sociocybernetic Framework for the Feature Analysis of Information Systems Design Methodologies, in [OLLE,83]

[IIKO,83] IIVARI, J and KOSKELA, E, An Extended EAR Approach for Information System Specification, in [DJNY,83]

[INTE,86] INTECH, EXCELERATOR 1.6 Reference Manual, INDEX TECHNOLOGY INC., 1986

[ISA,84] Decision Support Systems - The Executive Tool of the Future, Position Paper to ISA 84, Author Anonymous, 1984

[ISAD,84] ISADWP BCS, Information Systems Development: A Flexible Framework, the Journal of Development of the ISAD working party of the BCS Database Specialist Group, 1984

[JACK,75] JACKSON, M, Principles of Program Design, Academic Press, London, 1975

[JACK,83] JACKSON, M, System Development, Prentice Hall International, London, 1983

[JMA,85] JAMES MARTIN ASSOCIATES, Information Engineering Glossary, James Martin Associates, 1985.

[JMA,86a] JAMES MARTIN ASSOCIATES, Information Strategy Planning Handbook, James Martin Associates,1986

[JMA,86b] JAMES MARTIN ASSOCIATES, Business Area Analysis Handbook, James Martin Associates,1986

[JOHN,84] JOHNSON, P E, The Expert Mind: a new challenge for the information scientist, in Beyond Productivity: information systems for organisational effectiveness, (ed) T Bemelmans, North Holland, 1984

[JOMA,80] JONES, S and MASON, P J, Handling the Time Dimension in a Data Base, Proc First International Conference on Data Bases, Heyden, 1980

[KENT,78] KENT, W, Data and Reality, North Holland, 1978

[KENT,83] KENT, W, Fact-Based Data Analysis and Design, in [DJNY,83]

[KIDD,85] KIDD, A, Knowledge Elicitation - a pragmatic approach, Presentation to the British Computer Society Expert System Specialist Group, 1985

[KLKO,74] KLIMBIE, J W & KOFFEMAN, K I, Data Base Management, North Holland, 1974

[KLOP,83] KLOPPROGGE, M R, TERM: An Approach to Include the Time Dimension in The Entity-Relationship Model, in [CHEN,83]

[KOWA,79] KOWALSKI, R, Logic for Problem Solving, North Holland, 1979

[KUNG,83] KUNG, C H, An Analysis of Three Conceptual Models With Time Perspective, in [OLLE,83]

[LANG,63] LANGEFORS, B, Some Approaches to the Theory of Information Systems, BIT 3, 1963

[LANG,66] LANGEFORS, B, Theoretical Analysis of Information Systems, Studentlitteratur, Lund, 1966

[LANG,80] LANGEFORS, B, Infological Models and Information User Views, Information Systems 5, 1980

[LEE,78] LEE, B, Introducing Systems Analysis and Design, Vols I & II, National Computer Centre, Manchester, 1978

[LEGE,78] LEE, R M and GERRITSEN, R, Extended Semantics & Generalisation Hierarchies, Proc ACM SIGMOD International Conference on the Management of Data, Austin, Texas, 1978

[LIND,79] LINDGREEN, P, Event Directed Program Execution From Declarative Specifications, in [SCHN,79]

[LONG,82] LONGWORTH, G, Effective Systems Development: how to recognize and avoid the problems of software development, Butterworth, 1982

[LUND,79a] LUNDEBERG, M, GOLDKUHL, G and NILSSON, A, A Systematic Approach to Information Systems Development -I. Introduction, Information Systems, Vol 4, 1979

[LUND,79b] LUNDEBERG, M, GOLDKUHL, G and NILSSON, A, A Systematic Approach to Information Systems Development-II. Problem and Data Oriented Methodology, Information Systems, Vol 4, 1979

[LUND,82] LUNDEBERG, M, The ISAC Approach to Specification of Information Systems and its Application to the Organisation of an IFIP Working Conference, in [OLLE,82]

[LUND,83a] LUNDEBERG, M, Some Comments on the ISAC Approach in Connection with the CRIS-2 Papers, Position Paper to the CRIS 2 Conference, 1983

[LUNDB, 83] LUNDBERG, B, On Correctness of Information Models, Information Systems, Vol 8, No 2, 1983

[MACD,82] MACDONALD, I G, Systems Development in A Shared Data Environment - the Information Engineering methodology, in [BAKER,82]

[MACD,84] MACDONALD, I G, Information Engineering - a methodology to match fourth generation tools, in Application Development Tools, (ed) E E Tozer, Pergamon Infotech, Maidenhead, England, 1984

[MACP,82] MACDONALD, I G and PALMER, I R, System Development in a Shared Data Environment: the D2 S2 methodology, in [OLLE,82]

[MADD,78] MADDISON, R, (ed), Data Analysis for Information System Design, Conference Proceedings, British Computer Society, 1978

[MADD,83] MADDISON, R N, et al, Information System Methodologies, Wiley Heyden, 1983

[MAEHL,62] McCARTHY, T, ABRAHAMS, P W, EDWARDS, D J, HART, T P and LEVIN, M I, The Lisp 1.5 Programmer Manual, MIT Press, Cambridge (Mass), 1962

[MAFI,81] MARTIN, J and FINKELSTEIN, C, Information Engineering, Savant Institute, Carnforth, Lancashire, England, 1981.

[MALIK,83] MALIK, R, Let the Dataflow in a Machine Like Alice, Computing, Nov 24, 1983.

[MART,82a] MARTIN, J, Managing the Data Base Environment, Prentice Hall Inc, Englewood Cliffs, NJ, USA, 1982

[MART,82b] MARTIN, J, Application Development Without Programmers, Prentice Hall Inc, Englewood Cliffs, NJ, USA, 1982

[MART,84] MARTIN, J, An Information Systems Manifesto, Prentice Hall Inc, Englewood Cliffs, NJ, USA, 1984

[MART,85a] MARTIN, J, Action Diagrams, Prentice Hall Inc, Englewood Cliffs, NJ, USA, 1985

[MART,85b] MARTIN, J, Recommended Diagramming Standards for Analysts and Programmers: a basis for automation, Savant Institute, Carnforth, Lancashire, England, 1985

[McDE,82] McDERMOTT, J, R1: A rule-based configurer of computer systems, Artificial Intelligence, Vol 19, 1982

[McJA,82] McCRACKEN, D D and JACKSON, M A, Life-Cycle Concept Considered Harmful, ACM SIGSOFT Software Engineering Notes, Vol 7, No 2, 1982

[MICH,80] MICHIE, D, Expert Systems, The Computer Journal Vol 23, No 4, 1980

[MILL,85] MILLER, D, Strategic Data Analysis, in [HOLL,85]

[MITC,85] MITCHELL, W, Information Engineering, in [HOLL,85]

[MLH,78] MUMFORD, E, LAND, F and HAWGOOD, J A, Participative Approach to Computer Systems, Impact of Science on Society, Vol 28, No 3, 1978

[MOREY,82] MOREY, R C, Estimating and Improving the Quality of Information in a MIS, CACM, Vol 25, No 5, 1982

[MUMF,85] MUMFORD, E, Defining System Requirements to Meet Business Needs: a case study example, The Computer Journal, Vol 28, No 2, 1982

[MURT,83] MURTAGH, F, A Survey of Recent Advances in Hierarchical Clustering Algorithms, The Computer Journal, Vol 26, No 4, 1983

[MURT,85] MURTAGH, F, A Survey of Algorithms for Continguity- Constrained Clustering and Relating Problems, The Computer Journal, Vol 28, Nol, 1985

[NIJS,77] NIJSSEN, G M, (ed), Architecture and Models in Data Base Management Systems, North Holland, 1977

[OLLE,82] OLLE, T W, SOL, H G, and VERRIJN-STUART, A A, (eds), Information Systems Design Methodologies: a comparative review, North Holland, 1982

[OLLE,83] OLLE, T W, SOL, H G and TULLY, C J, (eds), Information Systems Design Methodologies: a feature analysis, North Holland, 1983

[ORMAN,83] ORMAN, L, Information Independent Evaluation of Information Systems, Information & Management, No 6, 1983

[PALM,78] PALMER, I, Experience with a Formal Methodology for Data Analysis, in [MADD,78]

[PARK,82] PARKIN, A, Data Analysis and System Design by Entity-Relationship Modelling: a practical example, The Computer Journal, Vol 25, No 4, 1982

[PETE,77] PETERSON, J L, Petri Nets, ACM Computing Surveys, Vol 9, No 3, 1977

[PROW,80] PROWSE, P, The Data Base Approach, The Computer Journal, Vol 23, No 1, 1980

[RIAL,78] RIDDLE, W E, WILEDEN, J C, SAYLER, J H, SEGAL, A R and STAVELEY, A M, Behaviour Modelling During Software Design, IEEE Transactions on Software Engineering, Vol SE-4, No 4, 1978

[RIDL,79] RIDDLE, W E, An Event-Based Design Methodology Supported by DREAM, in [SCHN,79]

[RIDU,82] RICHTER, G and DURCHOLZ, R, IML-Inscribed High-Level Petri Nets, in [OLLE,82]

[ROBI,79] ROBINSON, KA, An Entity/Event data Modelling Method, The Computer Journal, Vol 22, No 3, 1979

[ROEV,81] ROCK-EVANS, R, Data Analysis, IPC Business Press, Surrey, 1981

[RORI,82] ROLLAND, C and RICHARD, C, The Remora Methodology for Information Systems Design and Management, in [OLLE,82]

[ROSE,82] ROSENQUIST, C J, Entity Life Cycle Models and Their Applicability to Information Systems Development Life Cycles: a framework for information systems design and implementation, The Computer Journal, Vol 25, No 3, 1982

[ROSS,77] ROSS, D, Structured Analysis (SA): a language for communicable ideas, IEEE Transactions on Software Engineering, SE-3, No 7, 1977

[RTW,82] RZEVSKI, G, TRAFFORD, D B and WELLS, M, The Evolutionary Design Methodology Applied to Information Systems, in [OLLE,82]

[SAKAI,83] SAKAI, H, A Method for Entity-Relationship Behavior Modelling, in [DJNY,83]

[SCHN,79] SCHNEIDER, H J, (ed), Formal Models & Practical Tools for Information Systems Design, North Holland, 1979

[SENKO,75] SENKO, M, Information Systems: records, relations, set, entities and things, Information Systems 1, 1975

[SERN,81a] SERNADAS, A, SYSTEMATICS: its syntax and semantics as a query language (1), The Computer Journal, Vol 24, No 1, 1981

[SERN,81b] SERNADAS, A, SYSTEMATICS: its syntax and semantics as a query language (2), The Computer Journal, Vol 24, No 2, 1981

[SHAVE,81] SHAVE, M J R, Entities, Functions and Binary Relations: steps to a conceptual schema, The Computer Journal, Vol 24, No 1, 1981

[SHOR,76] SHORTLIFFE, E, Computer-Based Medical Consultation: MYCIN, North Holland, 1976

[SIZER,82] SIZER, D, Managing Information as a Corporate Resource, Computer Bulletin II/33, September 1982

[SMIT,77a] SMITH, J M and SMITH, D C P, Database Abstractions: aggregation, CACM, Vol 20, No 6, 1977

[SMIT,77b] SMITH, J M and SMITH, D C P, Database Abstractions: aggregation & generalisation, ACM Transactions on Database Systems, Vol 2, No 2, 1977

[SOMO,81] SOMOGYI, E, System Development Methods, Butler Cox, 1981

[SOWA,84] SOWA, J F, Conceptual Structures: information processing in mind and machine, Addison Wesley, 1984

[STAM,77] STAMPER, R, Physical Objects, Human Discourse and Formal Systems, in [NIJS,77]

[SUND,74] SUNDGREN, B, Conceptual Foundation of the Infological Approach to Data Base, in [KLKO,74]

[SW,84] Application Development in Practice, Software World, Vol 14, No 3, 1984

[SWBA,82] SWARTOUT, W and BALZER, R, On the Inevitable Intertwining of Specification and Implementation, CACM, Vol 25, No 7, 1982

[TAMA,85] TAMASSIA, R, New Layout Techniques for Entity Relationship Diagrams, in [IEEE,85]

[TBT,83] TAMASSIA, R, BATINI, C and TALAMO, M, An Algorithm for Automatic Layout of Entity Relationship Diagrams, in [DJNY,83]

[TEHE,77] TEICHROW, D and HERSHEY, E A, PSL-PSA: A Computer-aided Technique for Structured Documentation and Analysis of Information Processing Systems, IEEE Transactions in Software Engineering, Vol SE-3, No 1, 1977

[STAM,77] STAMPER, R, Physical Objects, Human Discourse and Formal Systems, in [NIJS,77]

[SUND,74] SUNDGREN, B, Conceptual Foundation of the Infological Approach to Data Base, in [KLKO,74]

[SW,84] Application Development in Practice, Software World, Vol 14, No 3, 1984

[SWBA,82] SWARTOUT, W and BALZER, R, On the Inevitable Intertwining of Specification and Implementation, CACM, Vol 25, No 7, 1982

[TAMA,85] TAMASSIA, R, New Layout Techniques for Entity Relationship Diagrams, in [IEEE,85]

[TBT,83] TAMASSIA, R, BATINI, C and TALAMO, M, An Algorithm for Automatic Layout of Entity Relationship Diagrams, in [DJNY,83]

[TEHE,77] TEICHROW, D and HERSHEY, E A, PSL-PSA: A Computer-aided Technique for Structured Documentation and Analysis of Information Processing Systems, IEEE Transactions in Software Engineering, Vol SE-3, No 1, 1977

[TI,86] TI, IEF Beta Test Manual, Texas Instruments, 1986

[TMHY,80] TEICHROW, D, MACASOVIC, P, HERSHEY, E A and YAMAMOTO, Y, Application of the Entity Relationship Approach to Information Processing Systems Modelling, in [CHEN 80]

-[TOML,84] TOMLINSON, T, Observations on Data Modelling in a Business Environment, Database and Network Journal, Vol 13, No 4, 1984

[TOZER,76] TOZER, E E, Database Systems Analysis and Design, Software Sciences Ltd, 1976

[TOZER,85] TOZER, E E, A Review of Current Data Analysis Techniques, in [HOLL,85]

[TSLO,82] TSICHRITZIS, D G and LOCHOVSKY, F H, Data Models, Prentice Hall Inc, Englewood Cliffs, NJ, USA, 1982

[VanM,79] Van MELLE, W, A Domain - Independent Production Rule System for Consultation Programs, Proc 6th International Joint Conference on Artificial Intelligence, 1979

[VEBE,82] VERHEIJEN, G M A and VAN BEKKUM, J, NIAM: An Information Analysis Method, in [OLLE,82]

[VERM,83] VERMEIR, D, Semantic Hierarchies and Abstraction in Conceptual Schema, Information Systems, Vol 8, No 2, 1983

[VERY,84] VERYARD, R, Pragmatic Data Analysis, Blackwell Scientific Publications, Oxford, 1984

[VonB,68] Von BERTALANFFY, L, General Systems Theory, George Braziller, New York, 1968

[WEST,81] WELTY, C and STEMPLE, D W, Human Factors Comparison of a Procedural and Non-Procedural and Query Language, ACM Transactions on Database Systems, 1981

[WFP,83] WASSERMAN, A I, FREEMAN, P and PORCELLA, M, Characteristics of Software Development Methodologies, in [OLLE,83]

[WILS,81] WILSON, I R, Programs Without Programming, in Proc BCS 81, British Computer Society, 1981

[WILS,85] WILSON, L, JSD in Practice, in [HOLL,85]

[WIRTH,71] WIRTH, N, Program Development by Stepwise Refinement, CACM, Vol 14, No 3, 1971

[WHFI,82] WOOD-HARPER, A T and FITZGERALD, G, A Taxonomy of Current Approaches to Systems Analysis, The Computer Journal, Vol 25, No 1, 1982

[YOCO,75] YOURDON, E N, and CONSTANTINE, L L, Structured Design, Yourdon Press and Prentice Hall, 1975

[ZADEH,74] ZADEH, L A, Fuzzy Logic & its Application to Approximate Reasoning, Information Processing 74, North Holland, 1974

[ZAHN,83] ZAHNISER, R, Levels of Abstraction in the System Life Cycle, ACM SIGSOFT Software Engineering Notes, Vol 8, No 1, 1983

[ZELN,79] ZELNMANN, E, Human-like Knowledge Acquisition by Natural Language Understanding and Learning, in Human & Artificial Intelligence, (ed) F Klix, 1979

X
APPENDICES

Appendix X1 - Information Engineering and Other Concepts

X2 - Analysis Views and External Interactions

X3 - Modelling Concepts Considered as Predicates

X4 - Glossary of Terms

X5 - Details of Published Material

X6 - Diagrammer Listings

X7 - Initial Proposal

APPENDIX X1 INFORMATION ENGINEERING AND OTHER CONCEPTS

Please note that this part is taken from a paper I co-authored at JMA [FMM, 86], after my research was incorporated into Information Engineering. Therefore it represents some of the results of my research as well as describing Information Engineering.

X1.1 INTRODUCTION

The Information Engineering methodology [MACD,82],[MACD,84], [MART, 82b], [MART, 84], [MAFI, 81] has been developed by James Martin Associates in the period since 1982. It is a fully developed approach to systems development, addressing every stage in the system's life cycle and is being employed by many organisations throughout the world (eg. [MITC, 85]).

Information Engineering is now conceived as a basis for providing automated support to the development process [GIMA,85]. It is a methodology which is based on a set of tightly integrated techniques which provide information required by a common methodology data model. The model is the basis for an active encyclopaedia containing all information relevant to the development process. The techniques are all diagrammatic and provide the primary means of input to the encyclopaedia. The encyclopaedia in turn controls consistency and mapping

between techniques. Ultimately the contents of the encyclopaedia drive a system generator. The final result of the integration and automation of Information Engineering is therefore the production of better quality systems and substantial improvements in productivity.

One constant design aim of Information Engineering is the convergence of the modelling of data and activity. Past methodologies have tended to be either heavily data-oriented or process-oriented. Information Engineering, while being based on the 'data-centred' approach, attempts to strike a balance when modelling data and activity.

X1.2 INFORMATION ENGINEERING STAGES

X1.2.1 INTRODUCTION

Information Engineering projects are organised around the stage framework shown in figure X1.1. Being task oriented, the diagram is a useful means of bringing out the practical aspects of the methodology. It is also sufficiently general to enable the place of the techniques and tools to be seen clearly.

X1.2.2 BUSINESS STRATEGY PLANNING

This is not formally part of the methodology, since it is normally carried out by corporate management and planning staff, but it is recognised as a necessary precursor.

The end product of Business Strategy Planning will be a Business Plan, indicating overall business goals and strategies. This should also show the main business functions and organisational structure, and the objectives established for each function. It must be framed in sufficiently quantitative terms for business information needs and priorities to be inferred.

X1.2.3 INFORMATION STRATEGY PLANNING

An overview is taken of part or all of the enterprise in terms of its business objectives and related information needs, its

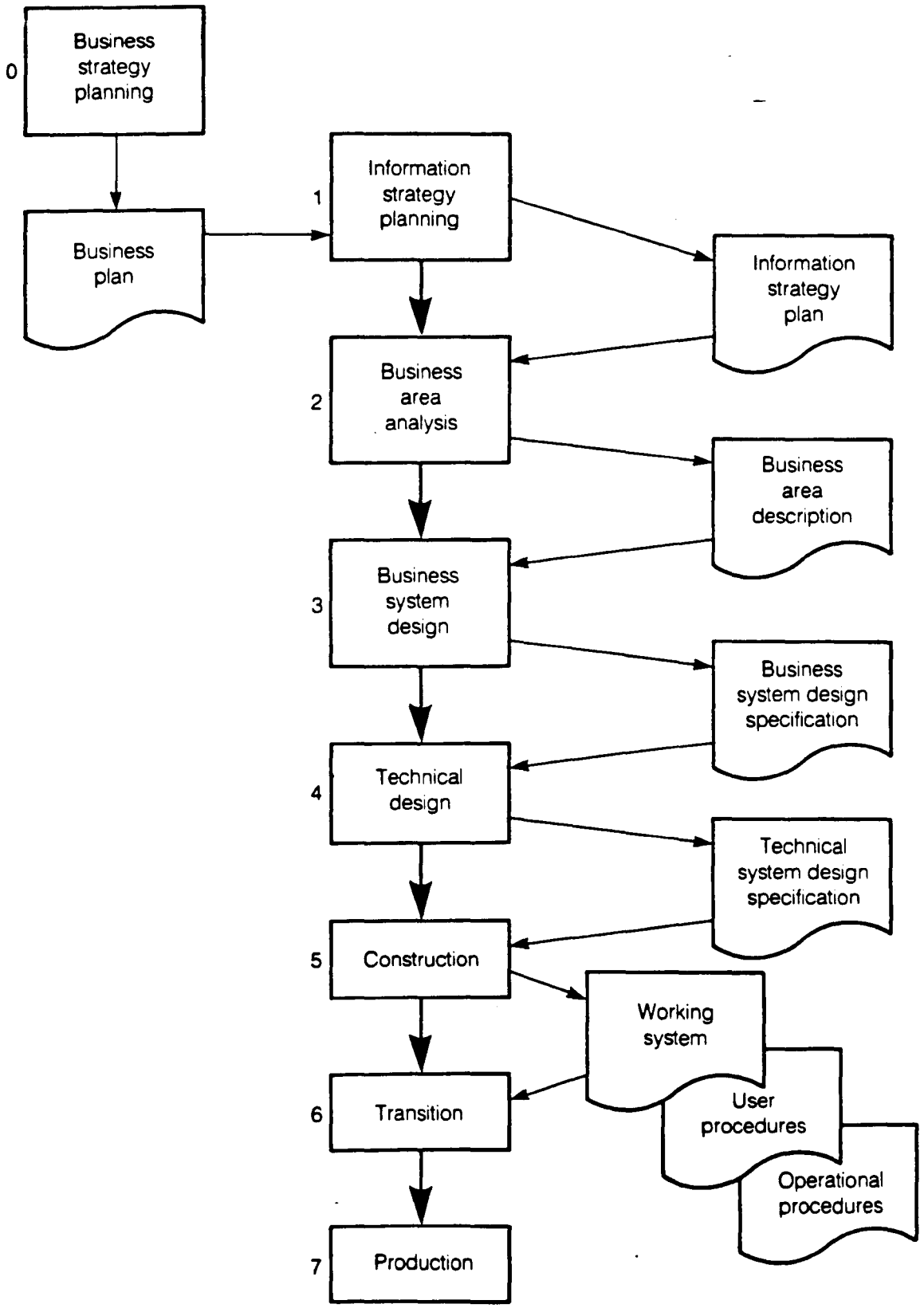


Figure X1.1 Information Engineering Stages

principal types of data (subject areas) and its business functions. This leads to the construction of an information architecture expressed in terms of a data_model (entity relationship model) and a function model (decomposition and dependency models). The architecture is ultimately divided into a number of business areas, each having the scope of a possible analysis project. The strengths and weaknesses of current systems are also assessed.

Based on these analyses a business systems architecture is drawn up and priorities for information systems development are set. The business systems architecture mainly consists of a set of existing, planned, and potential business systems and their interactions. A systems development strategy is produced which expresses priorities for addressing the systems needs of business areas.

The final product is an Information Strategy Plan which contains the information architecture and the business systems architecture. It also includes:

- a technical architecture which provides a statement of direction for the enterprise's computer hardware, software and communications facilities;

- a proposal for the organisation of the information systems function, to satisfy the demands of the strategy;

- a broad business evaluation;

- a migration path;

- a plan for systems development, including work programs for high priority projects.

Once the plan is complete, remaining issues should not be material to the strategic direction, ie. are unlikely to change the strategic plans. The architectures produced should be reasonably insensitive to the perceivable range of changes in business and technical plans, volumes and costs. The plan itself is likely to undergo periodic re-evaluation. The architectures will be refined through the carrying out of further Information Engineering stages, however their base content is likely to remain stable.

This approach enables the enterprise to establish the best

possible underlying systems and database architecture from which its developers can design and implement a coherent set of information systems. The strategy also establishes a context for re-evaluating assumptions and priorities so that it can be controlled, managed and revised at regular intervals.

X1.2.4 BUSINESS AREA ANALYSIS

For an identified business area within the scope of the Information Strategy Plan, a detailed study is carried out of its data, activities and all interactions between the two. This leads to identification of entity types and of the elementary business processes and their input and output information views. These are analysed in detail and their names, interactions, meanings, quantities, conditions and business algorithms documented. An important feature is that maximum involvement of end users in the specification of requirements, priorities and facilities is recommended.

At the end of Business Area Analysis, a Business Area Description is prepared, consisting of a Business Area Model showing business functions and their decomposition to the processes performed in the area, also entity types, relationships and attributes found in the area with their properties and their usage patterns in the business processes. This provides much greater detail than the information architecture and indicate information needs and priorities

within the area. The techniques used are entity relationship modelling, process decomposition and process dependency analysis.

From this information, a detailed statement of the business requirement for information systems in the area is produced.

It is then possible to identify the broad nature of likely computer support required for business processes and to define the scope of one or more business systems to be designed and a work program and resource estimates for them. All the information is present, about the business and its user's requirements, which is necessary to select particular business processes for computer support and to design the computer systems and the data structures needed to give that support.

X1.2.5 BUSINESS SYSTEM DESIGN

For the whole or a major part of a business area analysed, the facts gathered during analysis are used to design a system to meet the identified business requirements. The design includes all those parts of the system directly relevant to its users including transactions, dialogues and controls. It is kept as independent as possible of the technology to be employed in implementation. Prototyping techniques are used to replace many of the tasks traditionally gone through in this stage.

An important objective of this stage is that it should complete the system design as far as possible without pre-judging technical issues. It is also heavily user-oriented and requires agreement by these users on the ways in which they will interact with the system.

The final product from Business System Design is a Business System Specification showing, for each business process, the consolidated documentation of information flows and user procedures and, for each computer procedure, a consolidated and confirmed version of the results of Business Area Analysis, plus the dialogue design, screens, reports and other user interfaces and adjustments to the data usage patterns. From this, a detailed scoping of the intended computer systems is prepared together with a work program and resource estimates for the next stage.

Once this is done, all aspects of the system which relate directly to the user should be defined and stable, and sufficient information should exist to finalise estimates for, and to complete, technical design.

X1.2.6 TECHNICAL DESIGN

For the computerised aspects of the business systems specified above, the facts gathered during analysis are used to design those parts of the system which are dependent upon the computer

technical environment. This is carried out in sufficient detail for construction and operation to be adequately costed. This design includes logical and physical data structures, computer programs, operational procedures and interfaces. The level of detail in the design is dependent upon the selected implementation vehicle. Certain methods, eg. system generators, have much of the technical requirement pre-defined.

The aims during this stage are to define efficient computer systems to support the selected business processes and to develop good (plus or minus twenty percent) estimates of costs and timescales for construction and transition, in terms of manpower and computer equipment.

The end result of Technical Design is therefore a Technical Specification containing database designs and the application system technical design including batch runs, finalised conversation flows and definitions of programming work units. It also includes the technical architecture and standards for the system - the hardware and software environment selected, its mode of use and specific standards and conventions proposed.

Finally it identifies the content of the construction and transition stages and gives a work program and resource estimates for these stages.

This Technical Specification should provide a stable design

which meets the functional and performance objectives and is insensitive to likely business and technical changes.

In practice fourth generation tools such as system generators are enabling much of this stage to be automated. Only the design of operational procedures is not well supported although the use of tools introduces an element of standardisation to them.

X1.2.7 CONSTRUCTION

For each implementation unit identified during design, a system is put together. This includes installation of equipment, establishing files, setting up procedures and specifying, coding and testing programs. The aim in Construction is to develop an application system as defined in the technical specification which is on target as to timescale and budget, of an acceptable quality, and which contains all necessary operating and user procedures.

The Construction stage can be regarded as complete once the defined acceptance criteria for the application are met satisfactorily, covering: system functionality, stress testing, operational procedures, user interfaces.

A major aim in Information Engineering is to automate the Construction stage as much as possible. The most significant

part therefore becomes the testing since it confirms that the system generated does perform as the user expected. If not, alterations to the system are accomplished by code re-generation.

X1.2.8 TRANSITION

Transition is the phased replacement of existing procedures and files by the new system and data stores. it is governed by the Transition Plan, including a work program and resource estimates, which is normally finalised in parallel with the construction phase, although it is not really dependent on the outcome.

Transition can be regarded as successful when the system operates for a specified period within defined tolerances as regards performance, error rate and usability, and passes its post-implementation review.

X1.2.9 PRODUCTION

Production is the successful operation of the system, with tuning and modification as necessary, until eventually the Transition stage in some other project replaces the systems built in this project.

The main objectives in Production are to maintain service levels

and functional performance during the lifetime of the system and to respond promptly and effectively to changes in business requirements.

X1.3 NOMENCLATURE AND DIAGRAMMING CONVENTIONS

X1.3.1 NOMENCLATURE

Figure X1.2 illustrates the structure of the major objects in the methodology.

The objects are defined in the glossary (appendix X4).

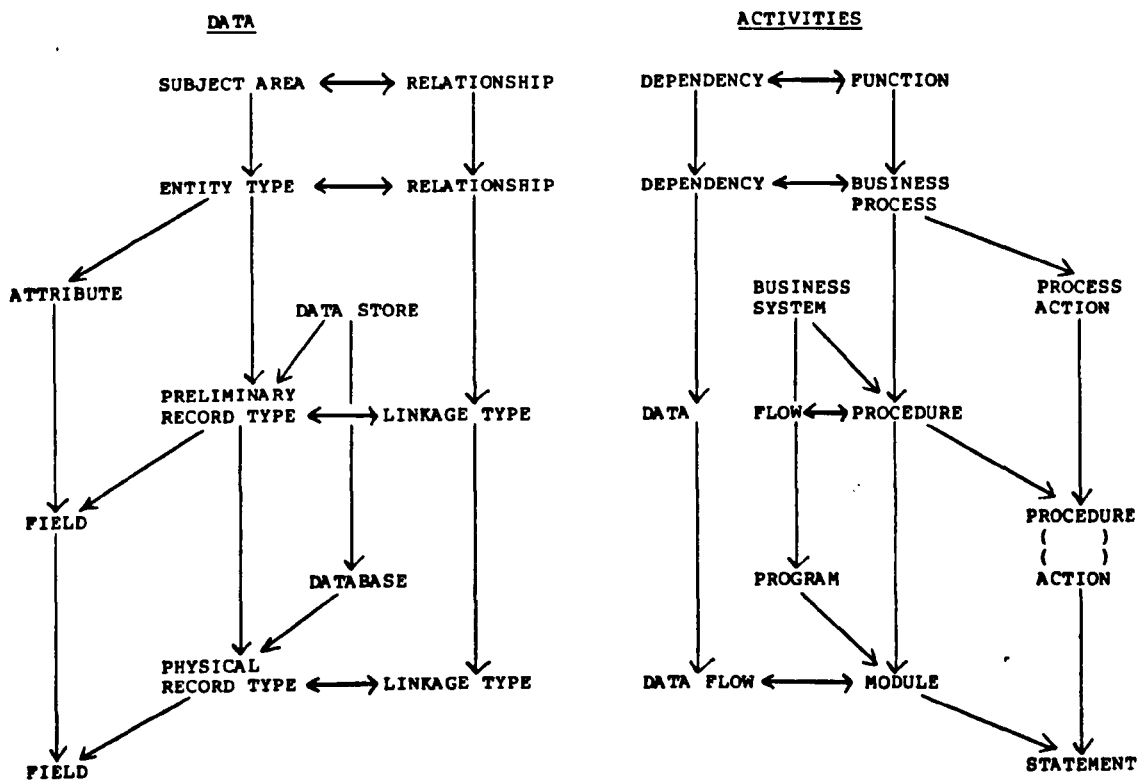


Figure X1.2 Major Object Types in the Methodology

X1.3.2 DIAGRAMMING CONVENTIONS

Diagrams are an important means of simplifying communication among those involved in Information Engineering projects [MART, 85a], whether end users or methodology specialists. Each technique is oriented towards diagramming and a diagram is delivered by every major task in the methodology.

Diagrams can be a most rigorous form of representation, particularly if it can be demonstrated that related diagram types can be transformed or mapped from one to the other without introducing inconsistencies. Accurate, controllable progression through the methodology is then assured and a basis for automation exists, with the encyclopaedia providing the information and mappings for each form of diagrammatic representation.

To achieve rigour in the diagrams and simplify their automation two features are stressed:

(i) the parallelism between ways of representing data and activities. This is discussed in greater depth in chapter D.

Diagrams with a similar purpose should have a similar style. A hierarchical decomposition of

data objects should look like a decomposition of activities since the principles of decomposition are always the same (figure X1.3). Likewise a diagram to represent associations between data objects (eg. an entity relationship diagram) should bear a strong resemblance to a diagram for representing associations between activities (eg. a process dependency diagram) as shown in figure X1.4.

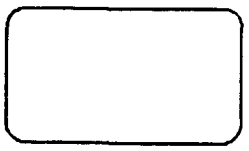
(ii) consistency of meaning in symbols (this is also discussed in chapter D).

Traditionally, methodologies have tended to stress differences between forms of representation. Information Engineering stresses similarities. A data object is always a rectangle irrespective of the stage in the methodology. An entity type therefore has the same visual significance as the record type to which it has been mapped so the viewer of the diagram need not be concerned with the mapping rules. Likewise a cardinality of many is always represented by a "crow's foot" irrespective of the type of diagram.

The principal conventions used in Information Engineering diagrams are as follows:

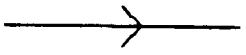
A rectangle represents a type of data eg. a subject area, entity type, record type, data store.

A soft box represents a type of activity
eg. a function, process, procedure, module or program.

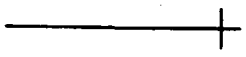


A plain line represents an association between objects
eg. a relationship or linkage.

A directed line represents an association between types of activity
eg. a dependency or data flow.



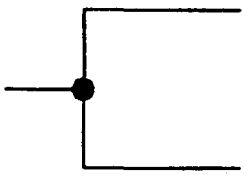
A small bar across a line represents a cardinality of one (rarely used).



A crow's foot on a line represents a cardinality of many.

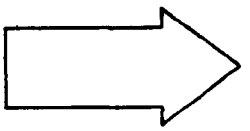
An O on the line represents optionality.

A solid circle at an intersection of lines represents mutual exclusivity among the associations leading away.



Note. Circles are always associated with a condition. The condition will cover either existence or execution.

An open arrow represents an event which triggers an activity.



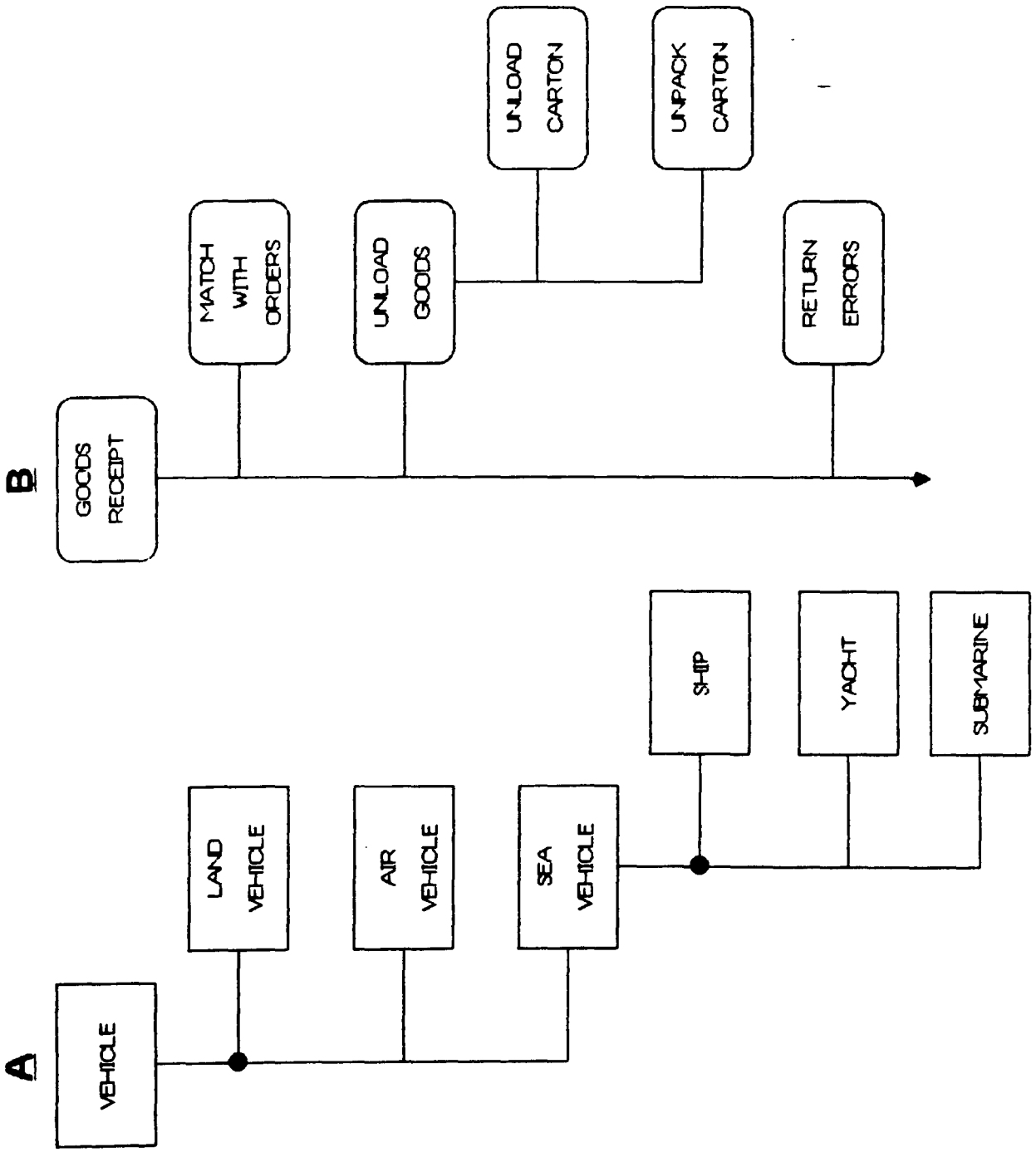


Figure X1.3 Decomposition of (a) an Entity Type (b) a Process

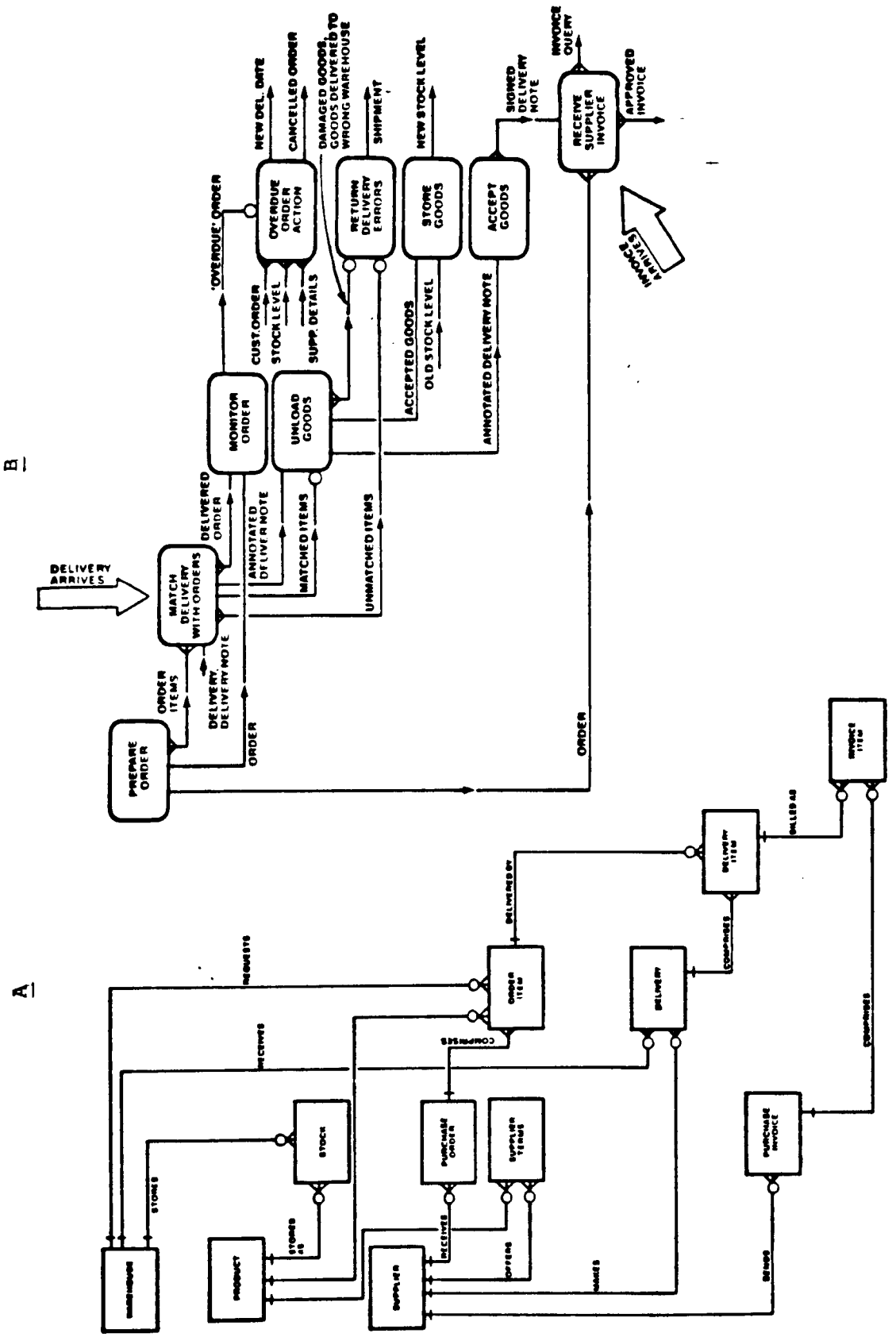


Figure X1.4 Association Between (a) Entity Types (b) Processes

X1.4 MACRO MODELLING

The main sets of diagrams employed in Information Engineering represent the principal objects identified by classifying features of the enterprise. These can either be decomposed into more detailed objects or, on any one level, the associations between objects of the same kind can be explored. Representations of these approaches are largely the same whether the objects are data or activities.

X1.4.1 DECOMPOSITION AND ABSTRACTION

Information Engineering is based around the application of abstraction and decomposition. Almost every technique contains some element of abstraction or decomposition, and every object can be abstracted from or decomposed into another recognisable object.

Abstraction is the process of distilling basic meaning from a set of objects while hiding detail. Decomposition is the reverse of this, ie. it is the process of finding detail from a less explicit object.

Three basic forms of abstraction have been found to be useful when applied to information modelling: classification, generalisation and aggregation [BRSI, 82], [SMIT, 77a], [SMIT, 77b].

Classification is a means of collecting together a set of individual objects into a meaningful higher-level object. The reverse of this is often called instantiation. An example of classification is to group together all the individual people in the world into a Person object. This is equivalent to a Person entity type with a collection of people entities.

Generalisation is the process of grouping a set of objects with some common properties into a higher-level object. The reverse of this is often called specialisation, ie. the taking of an object and decomposing it into a set of more specialised objects of a similar kind. For example, the set of Vehicles can be specialised into Land Vehicle, Air Vehicle and Sea Vehicle, which can themselves be further specialised. Generalisation is often characterised by use of an "is a" relationship.

Aggregation is a means of taking possibly disparate objects and forming a higher-level object from them. The reverse of this is known as individuation. An example of aggregation is to take the processes Decide Need, Decide on Supplier, Produce Purchase Order and abstract them to give a Purchase Ordering function. Aggregation is often characterised by a "part of" relationship.

One important property of abstraction and decomposition in general is that no information is lost or gained in the process, just hidden or exposed. When abstracting or decomposing, every object at one level must correspond to the objects at the other

level. When abstracting, a higher-level object must cover all the information of the lower-level objects with no information added.

When decomposing, the combination of all the lower-level objects must cover all that is implied by the higher-level object. Another important concept of abstraction is the inheritance of properties where lower-level objects inherit all of the values of the higher-level objects.

X1.4.1.1 Decomposition and Abstraction in Activities

As shown in figure X1.2, we identify three types of conceptual activity in Information Engineering, functions, processes, and process actions (or actions for short). Functions are broad activities which correspond to large parts of an organisation's activity, for example, Accounting, Employee Management and Customer Management. Processes are specific identifiable activities whose executions leave the organisation in a recognisable 'state', for example Taking an Order, Hiring an Employee and Producing Yearly Accounts. Actions are low-level activities on specific data with reasonably short execution times. Their execution does not necessarily leave the organisation consistent, for example, Check Customer Credit is often insufficient to leave the organisation 'consistent' on its own.

Activities are found through a process of top-down

decomposition. This is in fact individuation of lower-level activities which can be aggregated into the higher-level activity. Functions are decomposable into a collection of lower-level functions, or into processes, with dependencies between them. Similarly, processes are decomposable into a collection of lower-level processes, or into actions (see chapter B). The actions are decomposable into a collection of lower-level actions until there is a single action on an attribute. Processes and actions can be instantiated into particular executions at a particular time using a particular set of data. The execution of a process is just the execution of its constituent actions.

In general the decomposition of an activity is individuation into aggregatable activities as in figure X1.5a. However it is also possible to specialise some activities. For example, see figure X1.5b where Customer Processing may be specialised into Retail Customer Processing and Mail Order Customer Processing. Most often this will be useful where there is a similar specialisation in the corresponding data (eg. in figure X1.5a, Customer specialised into Retail and Mail Order Customers).

Decomposition of activity does not produce a strict hierarchy, as some activities may be part of a number of higher-level activities. It is important to find such processes to reduce redundancy in our models and resulting systems.

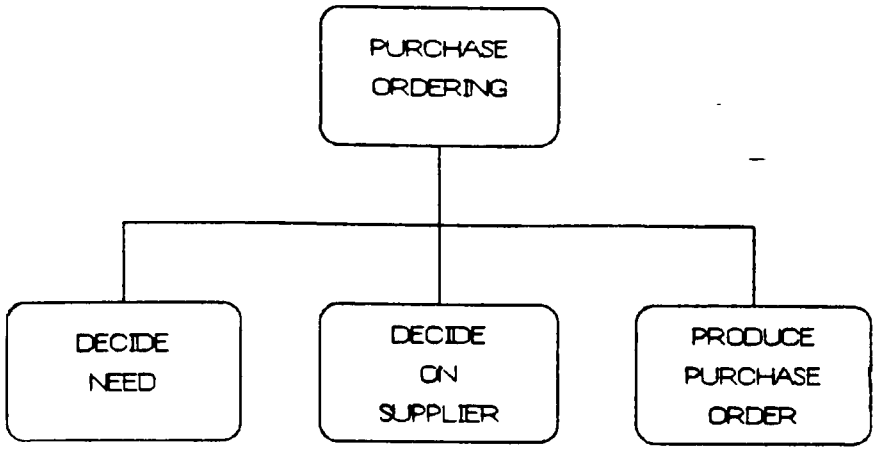


Figure X1.5a Individuation of Activity

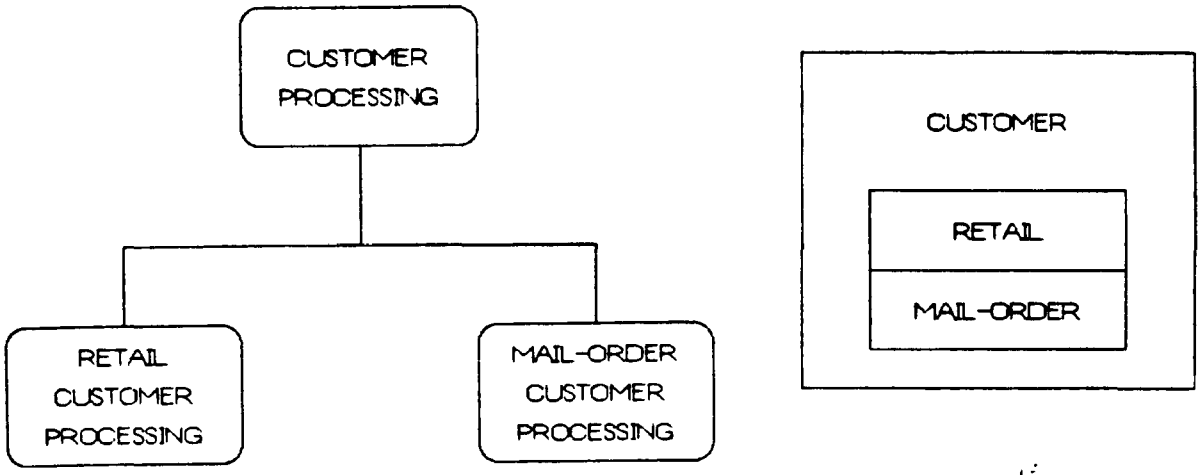


Figure X1.5b Specialisation of Activity & Corresponding Data

Figure X1.5 Decomposition of Activity

X1.4.1.2 Decomposition and Abstraction in Data

The use of the types of abstraction is most often described in the literature as applied to data. Thus, data can be decomposed into other data as shown in figure X1.6. At a single level (similar to functions being decomposed into processes) subject areas decompose into other subject areas or into a collection of entity types and their relationships. This is individuation and

the lower-level data is aggregatable into the higher-level subject area.

Evidence shows that the aggregation of entity types into subject areas produces something very close to a function [FEMI, 80]. There is one important exception: those entity types which are shared by a number of functions, and hence, appear in a number of subject areas. This suggests that there is little difference between functions and subject areas, with a subject area just being that data needed to support the function or alternatively that function which is needed to cope with the required subject area data.

Entity types can be specialised into subtypes (a subtype being a specific category of an entity type), which can themselves be further specialised as in figure X1.6b. Entity types have attributes which describe the specific properties of the entity type. An entity type can be decomposed into its attributes, which can be aggregated into the entity type (figure X1.6c). Also, the attributes and entity types can be instantiated (figure X1.6d); the occurrences of entity types, the entities, are classified into the entity types. This is one common way of finding entity types.

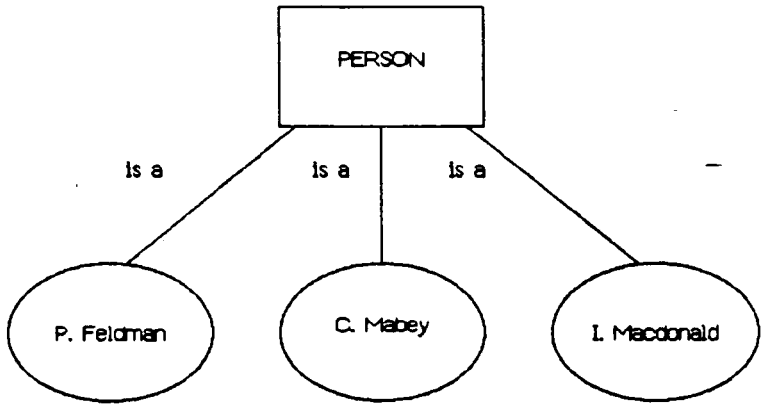


Figure X1.6a Instantiation of Data (Classification)

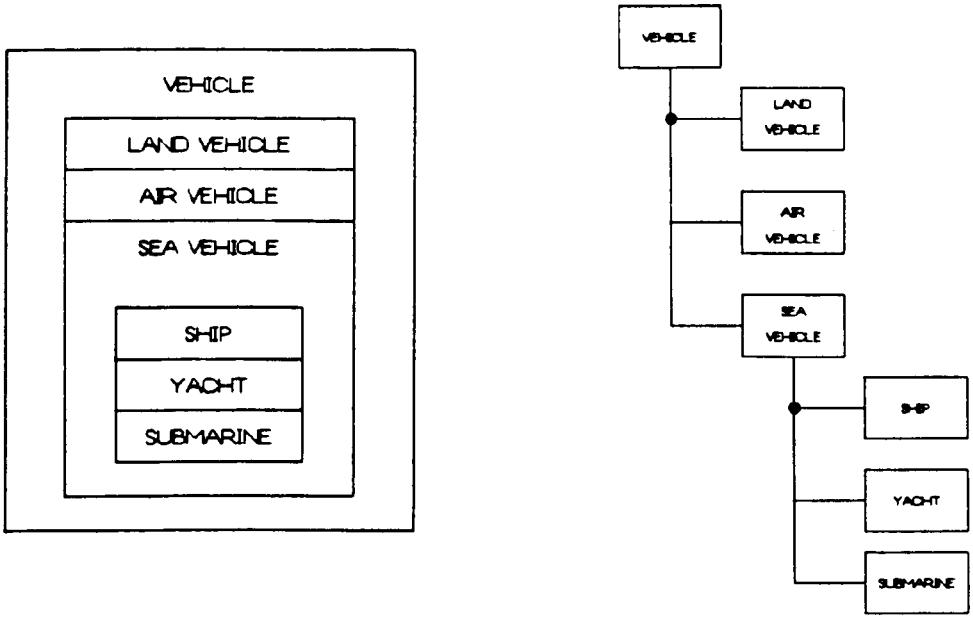


Figure X1.6b Specialisation of Data (Generalisation)

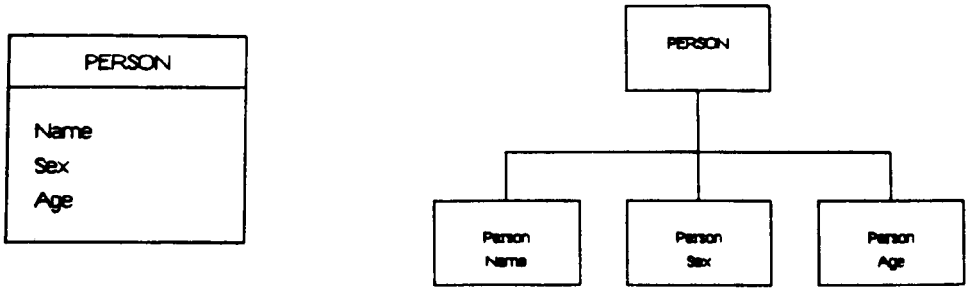


Figure X1.6c Individuation of Data (Aggregation)

Figure X1.6 Decomposition of Data

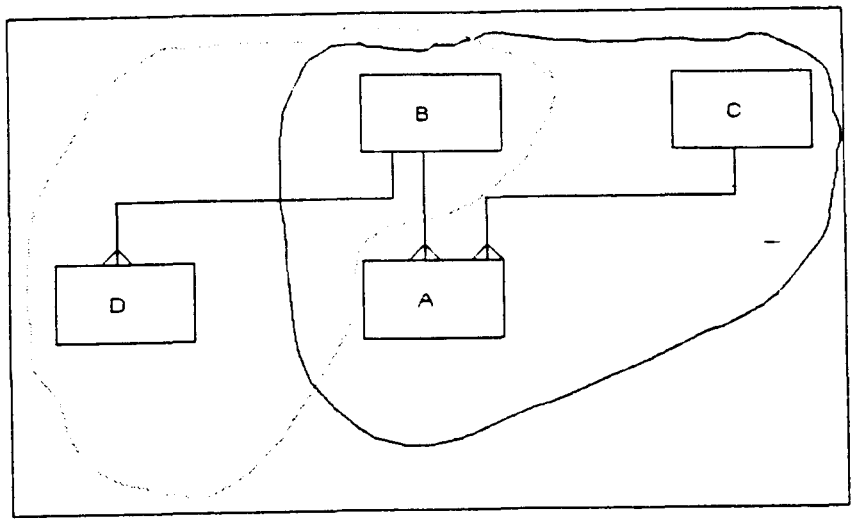


Figure X1.7a Logical Horizon of Entity Types A & D

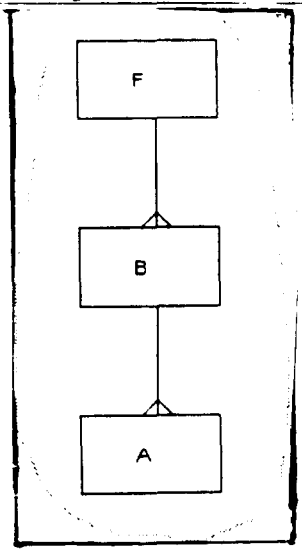


Figure X1.7b Abstraction of Logical Horizon

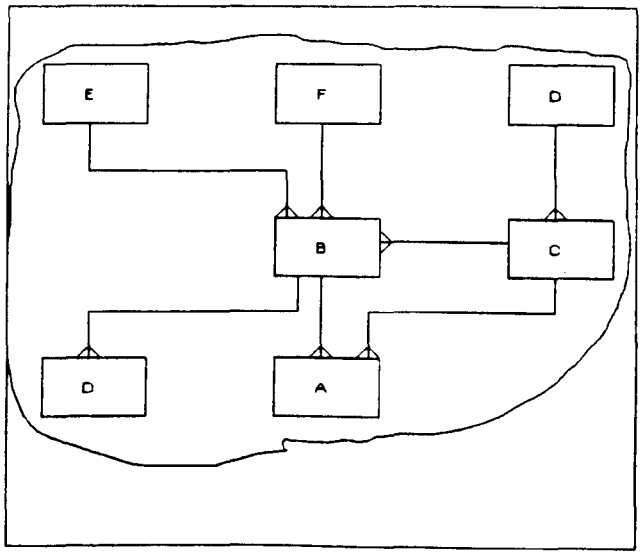


Figure X1.7c Subject Area as Logical Horizon of A (&D)

Figure X1.7 Logical Horizons and Subject Areas

As described by Ellis [ELLIS, 82], one form of aggregation abstraction applied to data is the concept of the logical horizon. A logical horizon is a grouping of associated entity types, the grouping arriving by virtue of the identifiability of entities of the entity types from one other (see figure X1.7a). Logical horizons are the aggregation of the entity types and their relationships into a grouping which can be aggregated with further horizons [ELLIS,82] (see figure X1.7). There is a fairly complex decomposition from subject areas to logical horizons as any one entity type may belong to a number of logical horizons. However a subject area should correspond to the highest-level logical horizon of the lowest level entity type, as in figure X1.7.

Some entity types belong to more than one subject area. So again, as for activity, the decomposition is not into a strict hierarchy. In fact one of the main advantages of entity relationship modelling is that it does produce a 'network', representing the actual structure within an organisation. The duplicated, or shared entity types are fundamental to the operation of the organisation. Such fundamental entity types are termed 'Major' entity types and are such things as Business, Person, Product and Organisation Unit.

X1.4.1.3 Summary

To summarise the above, functions are decomposed into functions or processes; processes are decomposed into processes or actions and actions can decompose into other actions. All can be aggregated and generalised to form higher-level activities. As a parallel, subject areas decompose into subject areas or logical horizons or entity types, logical horizons decompose into logical horizons or entity types, and entity types decompose into subtypes or attributes. All of these can also be aggregated or generalised to form higher-level data objects. All of these objects are part of a 'network', with the shared object often being fundamental to the operation of the modelled organisation.

Processes, actions, entity types and attributes can all be instantiated into occurrences (executions for activities), the occurrences being classifiable into the appropriate objects.

]X1.5 SEPARATING SPECIFICATION AND IMPLEMENTATION ISSUES

Current thinking is that analysis moves smoothly into design with a simple mapping of information. However when analysis is examined closely it can be seen that design often starts in the middle of analysis. The prime example of this is Process Logic Analysis, where the detail of processes is examined. Every methodology I have looked at which has an equivalent to Process Logic Analysis does this examination in a design-oriented manner. One of many possible solutions is taken for the inherent logic of a process and this is used to determine many things, for example, use of the entity relationship model. The designer has very little opportunity to question any decisions made, but many of these decisions are design decisions, for example, whether to delete an entity in a given process. (Incidentally action modelling solves this problem - see chapter B.)

There is a distinct difference between a business requirements decision and a design decision. For example, in a flight seat booking situation, all the business needs to know is that a flight seat is booked. It does not matter to the business (and hence to analysis) if a seat entity is created as booked or is created when the flight is planned and then marked as booked; the end result is the same. However when the plane is full it is a business requirements decision how to produce more seats - do you put on another flight (as in the shuttle services), expand

your aircraft, put on a larger aircraft, or just turn down the bookings? Normally both types of decision (business requirements and design) are dealt with in analysis!

As Swartout and Balzer point out [SWBA,82], it is never possible to totally dissassociate analysis and design. But we do need to produce as good a separation as possible. To achieve this I propose a new stage - Business Systems Analysis - which occurs between Business Area Analysis and Business System Design. Its purpose is to specify all the design-oriented considerations that are commonly dealt with by analysis, but after a system's requirements have been analysed in depth. These considerations are things which are habitually done in analysis, but I believe wrongly. Examples of these things are Process Logic Analysis as already discussed, volumetrics gathering, formats of attributes (eg. length, number of decimal places).

Additionally there are design issues which can be brought forward from Business System Design, for example, prototyping. The advantage to prototyping in Business Systems Analysis is that the entity types and processes with which a user is already familiar can be used; in Business System Design only record types and designed procedures are habitually discussed. Users are more likely to be cooperative and more able to participate if the fewest barriers are placed in their way.

APPENDIX X2 ANALYSIS VIEWS and EXTERNAL INTERACTIONS

An analysis of an area is affected by the scope defined for the area. This scope could be inter-departmental or even inter-enterprise. We will be seeing more of the latter in the future as different enterprises link their computer systems together. This raises the question of what actually 'belongs' in a business area and how it should be analysed.

It is possible to analyse an area with no discussion, just by an analyst's intuition. However this is not likely to be satisfactory. So an analyst needs to talk to the people who are working in a business area. As soon as this is done discussion and conflict will be generated as the people involved come to a single 'view' of the world. Any compromise is bound to miss some point, however minor, otherwise conflict would not have arisen. Ideally a view would be taken from every person and consolidated with a minimum of loss. Unfortunately this rarely happens. X2.1 discusses the issues which cause this conflict - the taking of analysis views.

Another area which affects analysis is what to do with things that are outside the scope. It is not satisfactory to just ignore them, as they impact on the area. This aspect is discussed in X2.2

The reason these discussions are included is because they affect the main research described in chapters B & C. In action modelling the area of events is due to external interactions and the actions depicted are determined by the views taken. In entity model clustering external objects invariably are major entity types. Also particular 'user' and departmental views can be constructed based on a consolidated model (see chapter C).

X2.1 ANALYSIS VIEWS

X2.1.1 INTRODUCTION

Business Area Analysis proceeds towards the goal of a stable representation of the content and structure of the data and activity in an organisation in the form of an entity relationship model. There will be a purpose behind this process, which is probably expressed as a set of 'terms of reference' and/or as a statement of purpose for any resulting information systems. It is generally felt that a 'business area model' should be free from bias towards particular structural and functional areas of the organisation, especially for the entity relationship model. In practise models are not free from bias and it is a daunting task to actually achieve this independence. The bias is not due to inherent problems in the structure of models, but arises from the process of gathering the underlying information and the construction of the model from it. For example, an entity relationship model produced for the ultimate purpose of forming an Order/Invoicing system will be heavily biased towards Orders, Invoices and their attendent information; any information which seemed extraneous to the analyst would not be shown. In a corporate-wide analysis (eg. as part of an Information Strategy Planning project) the bias will not be overtly functional, but be more subtle. In this case the bias is derived from the people interviewed, the information elicited and the analyst's attitude towards the

information considered to be most important, or least redundant. Most content and bias is either determined by the purpose of analysis, by the terms of reference, or by the analysts themselves. Thus a model can be said to give a view of the area under analysis; eg. the Order/Invoicing view, the view of analyst x at time y.

The main problems with views are that 'view models' lack the information that was not considered, lack the functionality that would exist in a 'view-free model', and the danger of a view not being prominently acknowledged on a model leading to misunderstandings and communication problems. It is debateable if it is possible to arrive at a view-free model, as an analyst will always reflect the source and purpose of information, but it is a desireable goal due to the problems of biased models.

There are grave implications for the development of further information systems if a full description is not available about models. For example, it is easy to take a complex view model as the basis for development of totally different systems to those the model was constructed for.

The problem of naming is minor and can be overcome without great difficulty. The problem of bias is deeper and more complex. The benefits of working by taking views, such as reduction of size of area under analysis and easier comprehension of familiar areas by a user, outweigh the benefits of some cures. The most

suitable solution is to provide a method of accomodating this method of working, while allowing for future flexibility by acknowledgement of the bias of the information presented. The following considers the different types of view and how they are manifested in models.

X2.1.2 TYPES OF ANALYSIS VIEW

This sub-section deals with two aspects of views, the types of view and how they affect entity relationship models in particular. There are at least three types of view, Purpose Views, Provided-Information Views and Extra-Dimensional Views; undoubtedly there are others. Most, if not all, analyses will have elements of all three types of view.

X2.1.2.1 Purpose Views

Purpose Views are views which are biased towards particular functional areas or organisational units such as Production, Order/Invoicing, Distribution. They arise from the purpose of analysis. Purpose views are common practise in large analyses where it is impractical to analyse a complete area at one time. They can be dangerous if not carefully controlled as they are based on the parts of an organisation with the highest potential for change, ie. they have a degree of inbuilt instability.

In a Purpose View the depicted information is determined by the

purpose of analysis. For an activity-based purpose (eg. the purpose is to produce an Order/Invoicing system) the information depicted will be the entity types and relationships needed to support the activity and the details of the sub-activities associated with the activity. Other information may be shown, but this either arises from other views, or is there for clarification. For example, classificatory entity types such as Type of Customer may be needed to clarify the information contained in other entity types, such as a Customer entity type. Similar arguments apply for a data-based purpose, eg. to produce a Customer Database. In this case the view of data is tightly restricted and the view of activity is even more restricted.

X2.1.2.2 Provided Information Views

Provided Information Views are views which are based on information gained from interview or some other means, and/or on an analyst's reasoning on this. Any analysis will result in a Provided Information View as the only information gathering techniques that exist rely on provided information. As Provided Information Views are unavoidable, the source of all information must be made clear and any biases inherent in the model should be spelt out. There are many dangers involved in not doing this as wrong judgements can easily be made. In practice an analyst would take a number of provided information views and synthesise their information content into a single view.

The content of a model must be determined by the information which is provided to an analyst, unless an analyst makes some assumptions. Assumptions are dangerous without some factual basis and corroboration, both of which would be provided information; ie. a corroborated assumption forms part of a Provided Information View.

X2.1.2.3 Extra-Dimensional Views

Extra-Dimensional Views are information which affect most, if not all, of a business area, but are on a different 'level' to the purpose of analysis. In effect this information gives another 'dimension' to a Purpose View. Examples of Extra-Dimensional Views are time and location of applicability.

Extra-Dimensional View information represents information that is necessary for a complete description of an area, but which cannot easily be dealt with, eg., time often surfaces in entity type names, as in Weekly Balance. The problem with Extra-Dimensional information is that it is prone to change. In fact most models are a static view of a dynamic area, thus there is an inherent time dimension to all the information to cater for the telescoping of its applicability.

The nature of information affected by an Extra-Dimensional View is such that it cannot easily be dealt with, as the information

belongs outside the main focus of analysis but is necessary to the analysis. Whilst often being to support the purpose of an analysis, the information is extra-dimensional to the purpose; eg. whether a balance is produced weekly or monthly is of little importance to the way it is produced or used, but could be important to its content and meaning. Ideally such considerations are generalised to the point where it does not matter.

Extra-Dimensional information affect a model in a number of ways, depending on the best way to handle the particular information and/or an analyst's preference. The outward affects of this type of view tend to be minimal. However Extra-Dimensional views can have far-ranging affects, especially on names chosen and the type of information recorded. By definition Extra-Dimensional Views can affect all of the information in an area.

X2.1.3 MANIFESTATION OF VIEWS ON INFORMATION

A typical analysis results in a Purpose, Provided Information view with several embedded Extra-Dimensional views. There are several ways the views affect a model. I will consider the case of an entity relationship model; process models are affected similarly. The ways dealt with here are the actual entity types shown, the relationships shown, the attributes chosen for an entity type, and the level of abstraction chosen.

X2.1.3.1 Entity Types Shown

In an entity relationship model the entity types provide a 'real world' representation, which is clarified by any relationships. For a particular area the actual entity types chosen and the names selected for those entity types will be determined by the view(s). Thus views manifest themselves in the base meaning of an entity relationship model. For example, take the case of analysing for a Decision Support System (DSS) in an oil company. An entity relationship model will be based on the information necessary to support the DSS, probably being most of the company. The names used will be those which are familiar to the ultimate users of the DSS, eg. Forecast Oil Revenue. Entity types that are quite important to the running of the company but are irrelevant to the DSS will be ignored, eg., Employees and Payments. Analysts will look at the company ' through the eyes of the DSS' and use those terms and entity types that they find support this view; these will be modified by the information provided to them and any extra-dimensional considerations that have to be dealt with, eg., Projected Weekly Oil Production.

X2.1.3.2 Relationships Depicted

There are two main types of relationships, those depicting a link caused by structure (structural relationship), as in Order

consists of Order Item, and those depicting usage in a common activity (functional relationships), as in Customer places Order. There are many possible relationships between any two entity types, but the communication potential of the model would be seriously impaired if all of them were shown on an entity relationship diagram. There are a number of remedies which address this problem. The most common is to show only those relationships that an analyst considers enhance the 'base' meaning. In fact this remedy is the imposition of views on the totality of relationships to abstract those relationships that meet the requirements of the views. The relationships depicted can only result from provided information. Of these relationships, only those which are useful to the purpose will be shown. Relationships between entity types are most useful as an aggregation over time of the interactions between entity types - ie. they show a static view of the interactions - so they are part of an Extra-Dimensional view. So all three types of view affect the relationships depicted.

Another remedy for dealing with a plethora of relationships is by not showing 'redundant' relationships (ie. relationships whose meaning is encompassed by other relationships). This is not an imposition of a view, but is a mechanism for improving the visual quality of a diagram. However views will affect the choice between two equivalent sets of relationships, probably by the rejection of the set of relationships that is least important to the purpose of a model.

X2.1.3.3 Attributes Chosen

The affect of views on the attributes chosen for an entity type is similar to the case of the entity types themselves, but attributes reflect views to a greater extent if anything. Attributes provide the base meaning of an entity type and reflect the actions on that entity type (most of the information required about an entity type is to support the way it is used). Thus attributes are affected by Purpose Views and can only be found from Provided Information Views. An entity types's attributes reflect any Extra-Dimensional Views, especially geographical and temporal, for example, most entity types will have a location and time of creation (date of birth).

X2.1.3.4 Level of Abstraction Chosen

Abstraction has long been recognised as a useful technique for aiding the comprehension and manipulation of data models (see A2.6 and appendix X1.4). The use of abstraction implies a hierarchy of data; the level chosen for this hierarchy determines the emphasis that is gained from an entity relationship model. The level of abstraction chosen is determined by the purpose which the entity relationship model is to be put to. Contrary to the other affects (ie. the affects on entity types, relationships and attributes), the use of

abstraction generally guides the analysis process to elicit the information needed to support it; ie. information is used to support an abstraction rather than vice-versa. However it must be remembered that all this information is provided by the area under analysis. Extra-Dimensional information only impinges on abstracted entity types through the other elements, it does not directly affect it.

X2.1.4 IMPORTANCE OF VIEW KNOWLEDGE

Apart from those benefits discussed in the preamble to this chapter, knowledge gained from the recognition of analysis views has a number of other benefits. These are a recognition of the source of information, easier update of information because its source and meaning are more easily determined, and a determination of the relevance of analysis information.

X2.2 EXTERNAL INTERACTIONS

External interactions are interactions with things external to the business area under analysis. An external object is one that cannot reasonably be affected by the modelled area itself, but which affects the modelled area, eg. Customer, Government, Head Office. In certain models a distinction is made between internal and external objects (eg. data flow diagrams [GASA,79] and dependency diagrams), but is hidden in entity relationship models where a Customer is treated as any other object for modelling purposes.

This difference arises from the nature of the information represented. In dependency models an emphasis of the model is on the interactions between the external and internal, whereas in an entity relationship model an emphasis is on the data needed by the internal about the external. In an entity relationship model internal and external data can be treated in essentially the same way, ie. there is no necessary difference between internal and external entity types; indeed it would be undesireable to create any difference. However in dependancy models there is a big difference between internal and external interactions as an external interaction implies a default system boundary which must be negotiated by the interaction. Also, any external decisions or replies to be made as a consequence of an interaction from a business area are outside the control of the business area and can only be based on the information sent out.

The definition of the scope of a business area could be inter-departmental or even inter-company. We will be seeing more of the latter in the future as different companies' computer systems are linked together.

APPENDIX X3 MODELLING CONCEPTS CONSIDERED AS PREDICATES

This appendix is extracted from [FMM,86]

X3.1 PREDICATES

A predicate is a mathematical way of representing an every-day language meaning or fact. It takes its name from the grammatical meaning of predicate ('the part of a sentence that expresses what is said of the subject'; 'a term designating a property or relation' [Webster]). For example, the sentence "Fred Jones is an analyst" consists of a subject, "Fred Jones", and a predicate, "is an analyst".

In mathematics the idea of a predicate is somewhat broader than this, allowing the subject to appear as a variable, for example:

is an analyst (PERSON)

The predicate is a function whose value is either true or false depending upon whether the value of Person, ("Fred Jones", "Tom Smith", etc) is actually an analyst or not.

The idea of a predicate can be extended to as many variables as necessary. This is denoted as follows:

$$P\ (X_1,\ X_2,\ X_3,\ X_4 \ldots\ldots X_n)$$

Predicates may be manipulated using the, so called, predicate calculus. Predicate calculus is the basis for relational database theory [CODD,70], is used extensively by artificial intelligence researchers and finds its expression in the Prolog language [CLME,81].

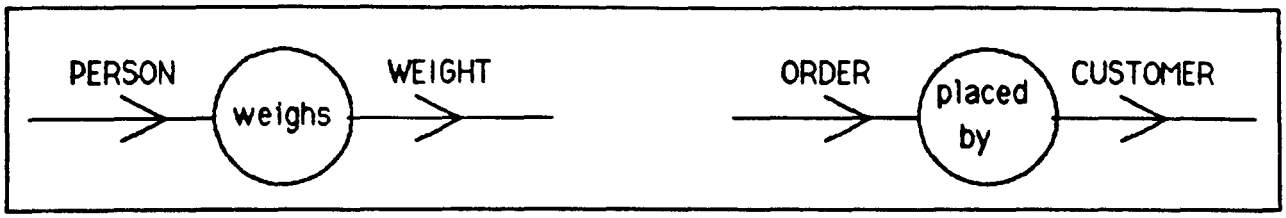
All the facts recorded in an entity relationship model can be expressed using predicates with only two variables, for example:

weighs (PERSON, WEIGHT)

placed by (ORDER, CUSTOMER)

The first corresponds to our notion of an attribute and the second to that of a relationship. Note that Weight, which we would normally regard as an attribute, has a similar status to Person, which we would normally regard as an entity type. Both are referred to as objects with no 'a priori' status. Giving both objects a value, eg. 'John' and '70 kilos' or '10593' and 'Acme Computer Ltd.', will result in a value which is true or false.

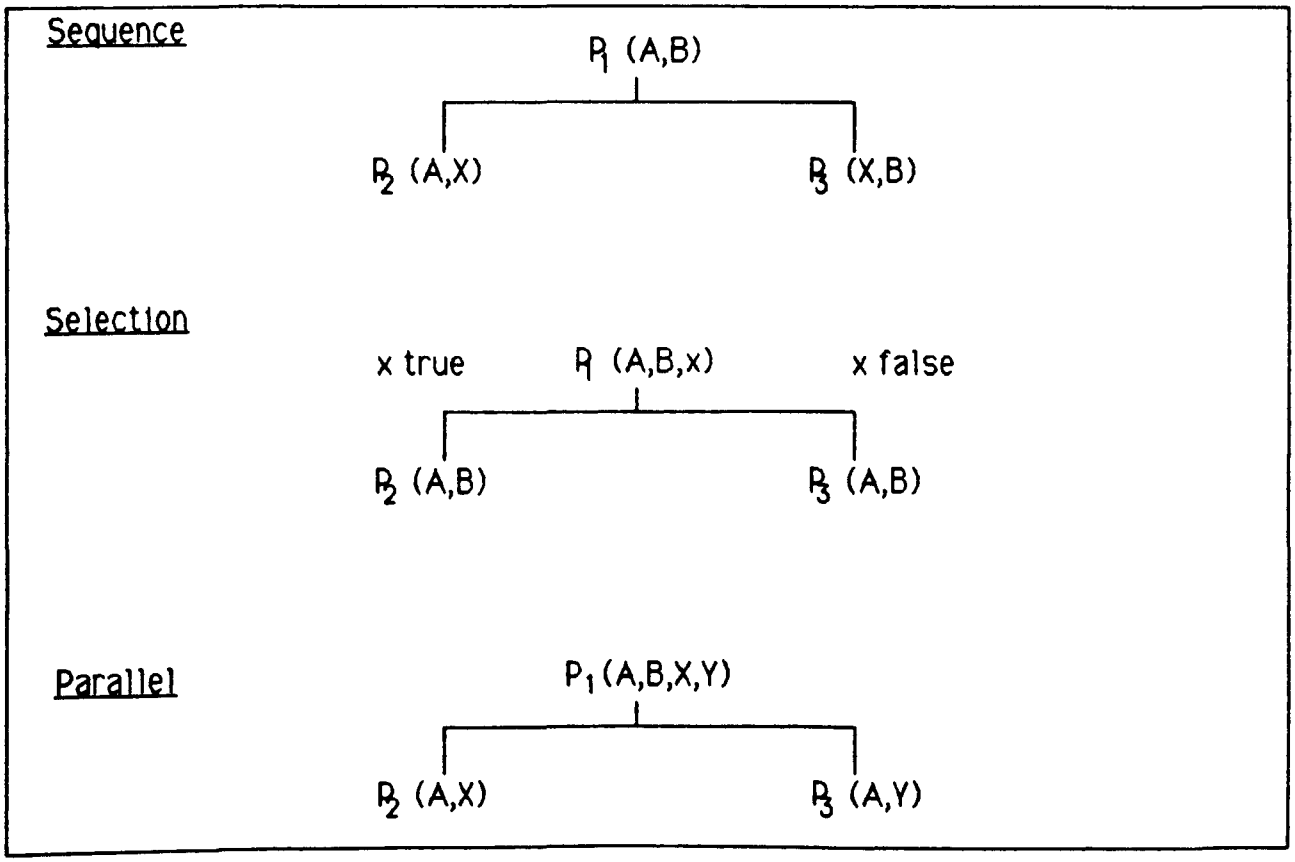
Another way of looking at these predicates is as functions whose input is the first argument and whose result is the second, thus:



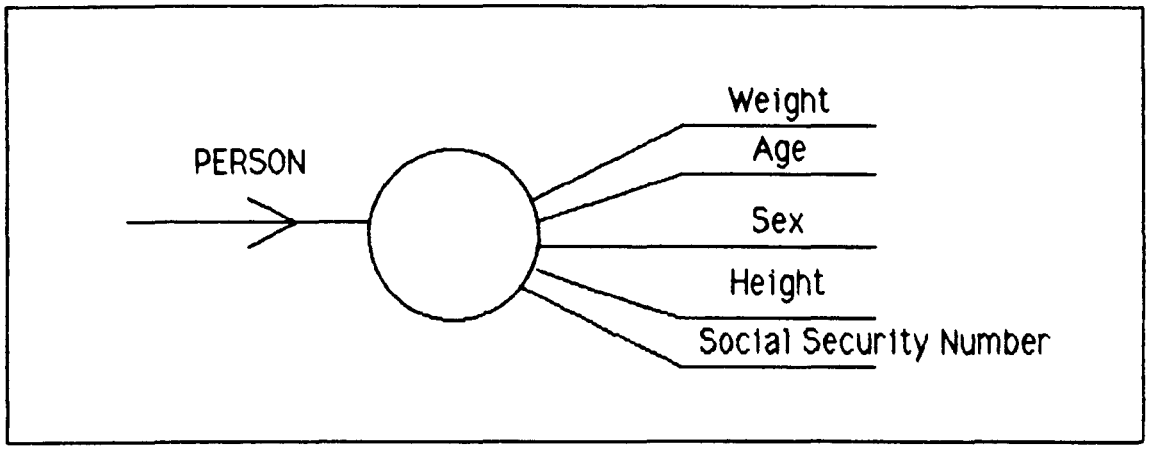
We also know that systems can be represented as a hierarchy of such functions using the three simple control structures from HOS. If this is the case we should be able to represent the entity relationship model in a symmetrical fashion.

X3.2 APPLYING ELEMENTARY CONSTRUCTS TO DATA

The HOS primitive control structures of JOIN (sequence), INCLUDE (parallel) and OR (selection), are interpreted in predicate terms below:

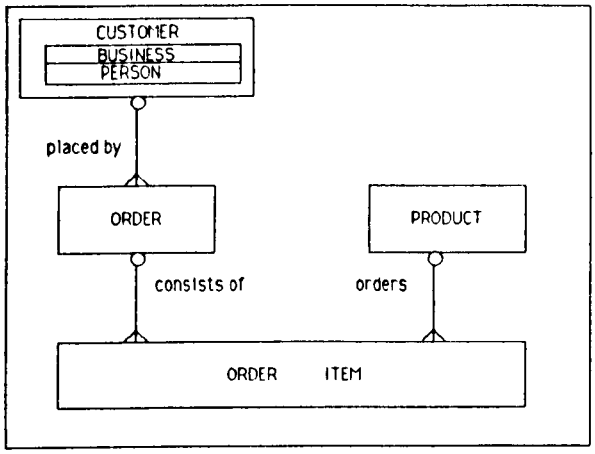


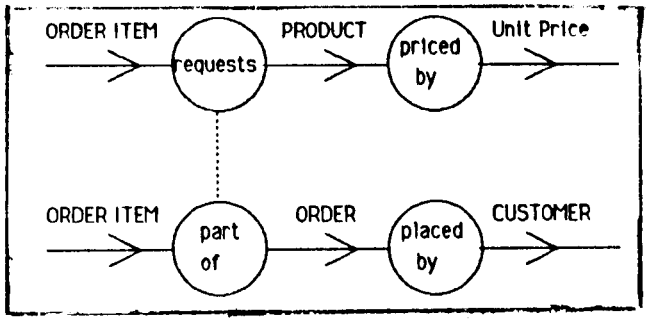
More complex structures with fewer restrictions can be built from these primitives. Note that higher level functions have progressively more arguments. At some point we may get something like the following:



This function will correspond to our notion of an entity type.

Figure X3.1 shows how a portion of an entity relationship model might be expressed in these terms. Only single valued predicates have been allowed. This is consistent with relational theory and has the repercussion that unresolved many to many relationships will not be expressed.





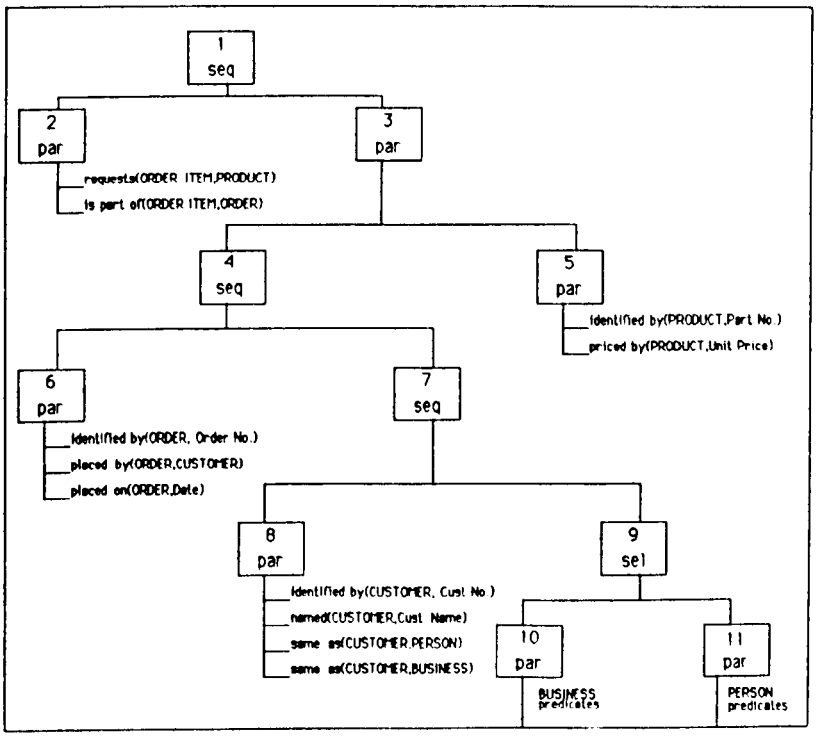


Figure X3.1 Entity Relationship Model Expressed by Predicates

An entity relationship model may be expressed in these terms but so might any other approach, such as bottom up normalisation or canonical synthesis [MAFI,81]. This model does permit the categorisation of objects into entity types, attributes, relationships and subject areas to be deferred. In some cases this is exactly what is required. On the other hand these ideas

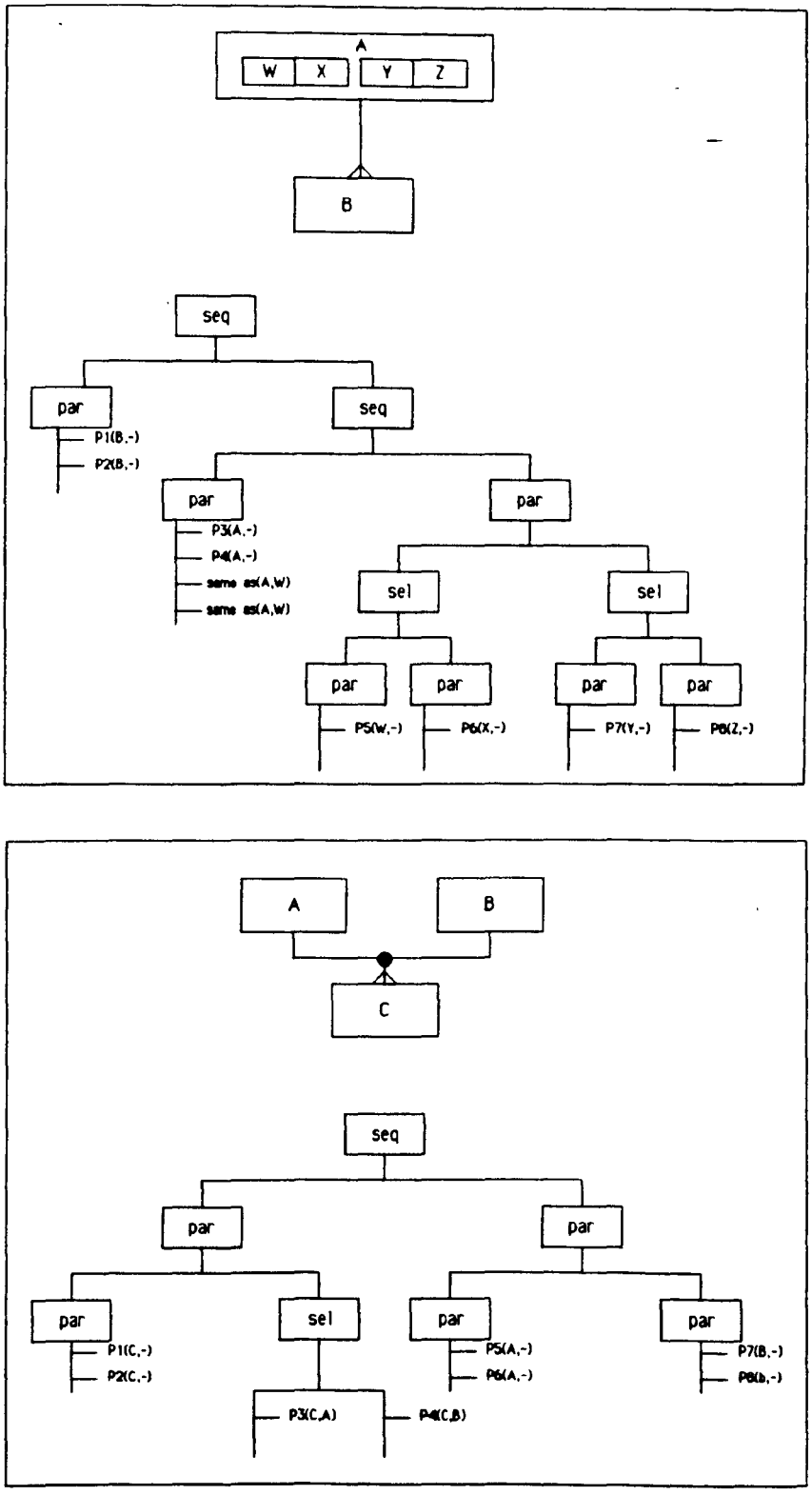
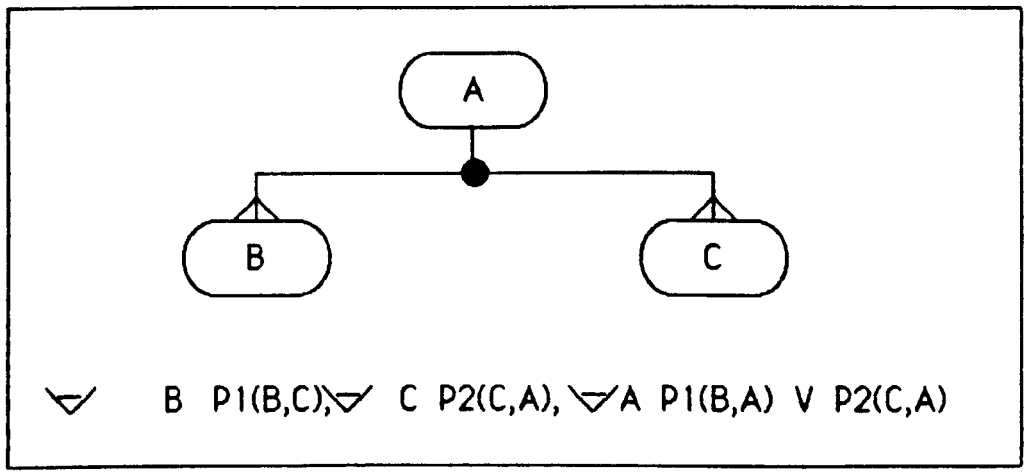


Figure X3.2 Entity Relationship Model Refinements Expressed by Predicates

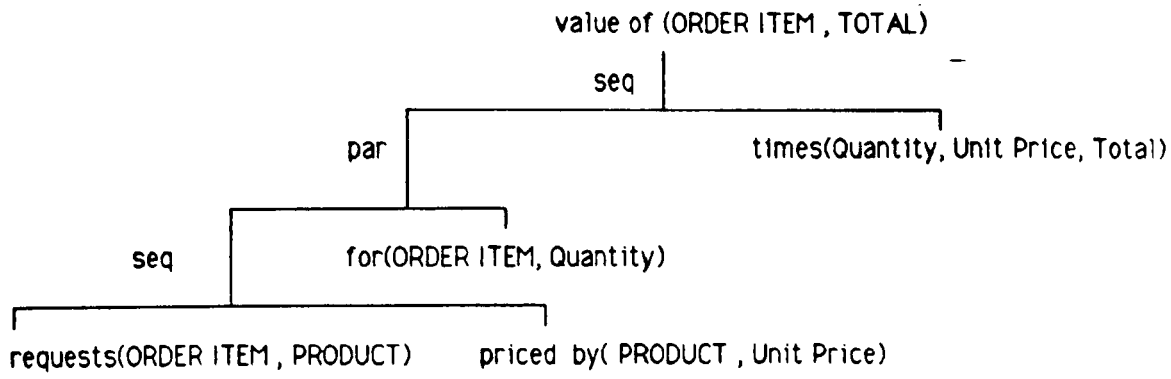
are quite simply expressed in these terms and could, indeed, be inferred automatically. So in figure X3.1 constructs 2, 5, 6, 8 are seen to be related to entity types and 10, 11 to subtypes, and so on.

Figure X3.2 shows how certain other constructs in an entity relationship diagram can be represented. The approach does not require the expression of optionality or exclusivity at the 'one' end of a relationship. Optionality is largely by default, but this is not very important since in most database management systems optionality is also the default. Mandatory relationships and exclusivity need to be expressed by integrity rules. Integrity rules can be recorded as predicate calculus statements, eg.,

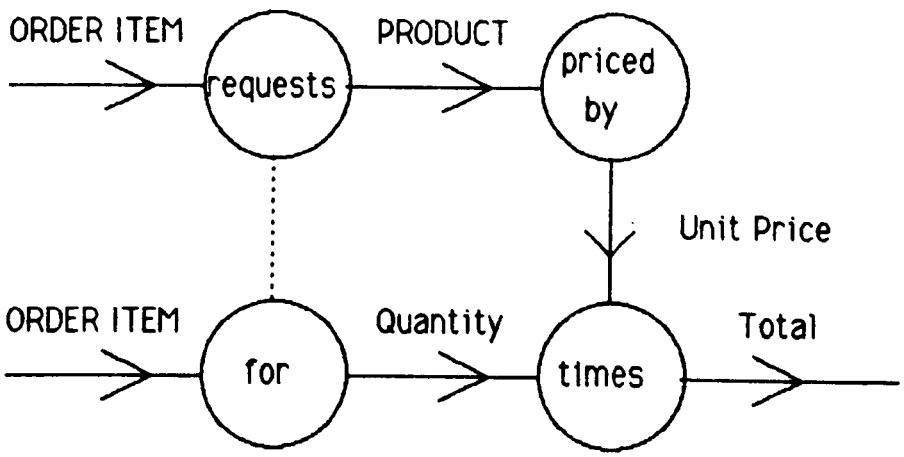


These would not be expressible in relational database terms anyway.

Derived attributes can also be represented as follows:



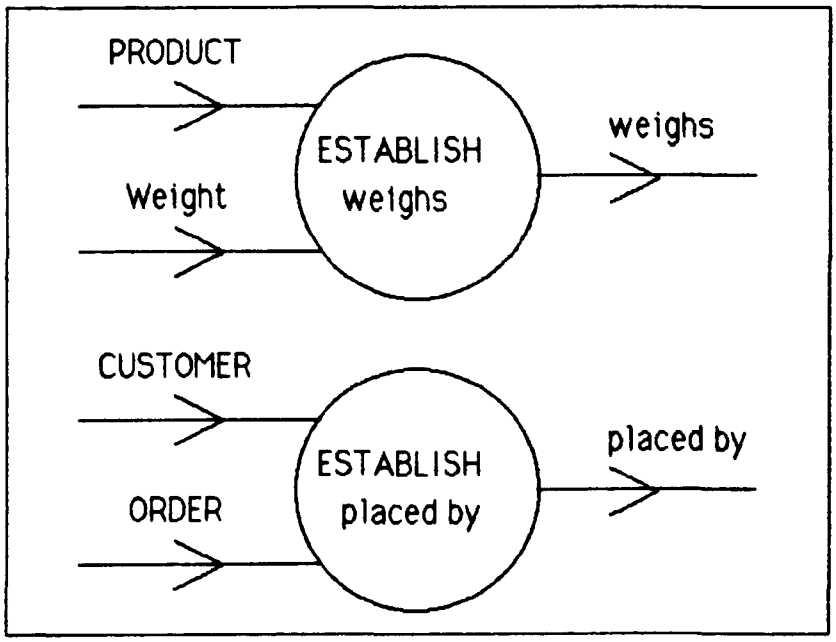
or expressed another way:



Note that a predicate can turn up in more than one place.

X3.3 SYMMETRY WITH ACTIVITIES

For each object defined in the data model there will be a mirror image in the process model. It's purpose will be to establish the predicate defined in the data model, thus:



In an implemented system the inputs would correspond to external inputs, say on a screen, and the output would correspond to the writing of one or more records.

Looking at things from the activity point of view every action can be thought of as a predicate, for example:

plan (x, y), where x & y are data objects.

In the entity relationship model we only consider those actions which are useful to associate the objects. We do not consider a whole lot of others. For example,

placed by (ORDER, CUSTOMER)

represents the basic association between Order and Customer. We still need to identify such actions as cancellation and modification, eg.

cancels (ORDER, CUSTOMER) and
updates (ORDER, CUSTOMER)

Each of these is dependent on the basic association having happened in the first place.

Because of the symmetry of the two structures it should be possible to stop decomposing at the elementary process level, or

the immediate levels below this (depending on content, context and recognisability of activity) and have the logic generated automatically and in its entirety by the system on the basis of incoming information views consisting of predicates.

APPENDIX X4 GLOSSARY OF TERMS

As of January 1985 this research was based on Information Engineering. As such the terminology of Information Engineering had to be used. Information Engineering has a reasonably comprehensive glossary [JMA,85], a lot of which has no relevance to this thesis. What follows is an extract from this glossary describing those terms which are referred to in this thesis. There are some additions to cater for the novel additions of my research.

ACTION
A type of activity by which an entity of a given type or a value of a given predicate is involved in a process,
OR A description of a particular use of an entity type

ACTION DEPENDENCY
An action is dependent on another action where the second action changes the state of an enterprise so that the first action can occur.

ACTION DIAGRAM
A representation of the logic of a process, a procedure or a module in terms of action statements (See Process Action Diagram).

ACTION ON ATTRIBUTE
A type of action performed on a value of an attribute during the execution of an action on an entity of a given type.

ACTION ON ENTITY TYPE
A type of action performed on one or more entities of an entity type at a particular point during the execution of an elementary process.

ACTION ON PAIRING
A type of action on one or more pairings of a given relationship by an elementary process following an action on an entity.

ACTION ON PREDICATE
See ACTION ON ATTRIBUTE, ACTION ON PAIRING.

ACTIVE ENTITY TYPE
An entity type whose entities are events or activities which take place.

ACTIVITY
An object which is a business system or function, or which can otherwise be executed.

AFFINITY (of one entity type to another)
The ratio of the number of processes using both entity types to the number of processes using the first entity type.

ANALYSIS AREA
See BUSINESS AREA

ANALYST
See BUSINESS AREA ANALYSIS

ANALYST
A person who works on the tasks of the Business Area Analysis or Information Strategy Planning stages.

APPLICATION PACKAGE
A commercially available software package which provides automated support to a business system.

APPLICATION SYSTEM
See BUSINESS SYSTEM

ASSOCIATE
An action by which an entity pairing is established or by which a record is added to a linkage.

ASSOCIATION
A pairing of objects.

ASSOCIATIVE ENTITY TYPE
An entity type whose entities relate to two or more entities with different roles.

ATTRIBUTE
A descriptor, values of which are generally associated with individual entities of a specific type.

ATTRIBUTE VALUE
A quantitative or descriptive characteristic of an entity.

AVAILABLE STATE
A state in the life of an entity, when it may be involved in, or transformed by processes of the enterprise.

BASIC ATTRIBUTE
An attribute whose values cannot be deduced or calculated and hence which must be collected during the execution of some process.

BEHAVIOUR PATTERN
A set of associated action occurrences as determined by the states of an enterprise's entity occurrences when the pattern is formed.

BENEFIT ANALYSIS
The study of the potential advantages to the enterprise, were selected processes to be supported by applying specific techniques.

BUSINESS AREA
A collection of business functions and entity types which are carried out together or are used together and which define the scope of an analysis project.

BUSINESS AREA ANALYSIS
The period in the systems life cycle in which a detailed analysis of business elements is carried out within a defined business area in preparation for the design of systems to support that area.

BUSINESS AREA ANALYSIS PROJECT
An activity established to analyze in detail a collection of closely related business functions and entity types.

BUSINESS AREA MODEL
The complete model of a business area; it consists of an entity relationship model, a process decomposition and dependency model, life cycle models and process logic models.

BUSINESS CHANGE
A possible or planned future event that will change the structure of the information architecture if it occurs.

BUSINESS FUNCTION
A group of business activities which together completely support one aspect of furthering the mission of the enterprise.

BUSINESS FUNCTION DEPENDENCY
An association between two business functions which exists because information provided by one is required by the other.

BUSINESS OBJECT
An object which forms part of the Information Architecture.

BUSINESS PROBLEM
A constraint which prevents the objectives of the person responsible for one or more activities being fully achieved.

BUSINESS STRATEGY PLANNING
The activity preceding the systems life cycle in which the objectives and strategies of the enterprise are set, so providing prime input to the Information Strategy Planning stage.

BUSINESS SYSTEM
An integrated collection of procedures which supports or is planned to support particular functions of the enterprise.

BUSINESS SYSTEM DESIGN
That part of the Design Stage during which those aspects of the system with which the user is directly concerned are specified.

BUSINESS SYSTEM DESIGN PROJECT
An activity established to design those aspects of a system with which the user is directly concerned.

BUSINESS SYSTEMS ARCHITECTURE
A structure expressed in terms of a data flow model, which represents the dependencies between the business systems of the enterprise and the data files which support them.

BUSINESS SYSTEMS PLANNING
A methodology for analysing the information handling needs of an enterprise, based on its overall objectives and the perceived needs of its existing systems.

CANONICAL SYNTHESIS
The construction of a normalized global data model by adding together two or more normalized local data models.

CARDINALITY OF A DEPENDENCY
The number of executions of each process that may occur prior to or subsequent to each execution of the other process.

CARDINALITY OF A RELATIONSHIP
The number of pairings in which an entity in one role may participate under the relationship.

CARDINALITY OF A SUBPROCESS
The number of times a subprocess is executed during each execution of the process of which it forms a part.

CLASSIFYING ATTRIBUTE
An attribute whose entities provide categories for entities of one or more other entity types.

CLASSIFYING ENTITY TYPE
An entity type whose entities provide categories for entities of one or more other entity types.

CLUSTER ANALYSIS
The grouping of objects of one type based on their commonality of involvement with objects of a second type.

COMPARISON CHECKING
A technique of completeness checking in which a model or design is compared with a model or design of appropriate current systems.

COMPLETENESS CHECKING
A procedure for confirming that the model or design produced by a project is complete in that every required element is present and defined.

COMPOSITE ATTRIBUTE
A named collection of attributes of one entity type, such that each group of values may be treated as if they were a single value.

COMPOSITE CLASSIFIER
Two or more attributes whose values, taken in combination, partition the entities of one type into entity subtypes.

COMPOSITE IDENTIFIER
Two or more attributes whose values taken in combination uniquely identify the entities of one type.

CONCLUSION
The inference that can be drawn from the results of an occurrence of an activity.

CONDITION
A rule expressed in terms of predicates and/or constants, which describes one aspect of the behaviour of the business.

CONDITION EVENT
A specific situation in the enterprise, occurrences of which trigger the execution of one or more processes.

CONDITION INVOLVEMENT
A situation whereby a condition is part of another condition.

CONDITIONAL MEMBERSHIP
The membership of an entity in a pairing where that membership depends upon its predicate values.

CONDITIONAL SUBPROCESS
A process whose execution depends on predicate values established by prior processes.

CONSTRUCTION
The period in the systems life cycle in which the systems to support a defined area are coded and proven according to the detailed design specifications produced during the Design Stage.

CONTROL CONDITION
A rule expressed in terms of logical data constructs and/or layouts and/or constants, which states when a procedure or step or action statement may or may not be executed.

COOPERATING PROCESSES
Two processes such that each is dependent on predicates being passed from the other.

CORRECTNESS CHECKING
A procedure for confirming that the model or design produced by a project is correct in that it is logically consistent and conforms to all standards specified.

COST/BENEFIT ANALYSIS
A technique for evaluating the business worth of a business system or a part of a business system.

COST/BENEFIT FACTOR
An aspect of business system development or usage which can be evaluated (not necessarily in monetary terms) as contributing to the costs or benefits of the system at any point in its life.

CRITICAL SUCCESS FACTOR
A result which is measurable and which will have a major influence on whether or not the organizational unit meets its objectives.

DATA
Organized facts and figures.

DATA ACCESS DIAGRAM
A map showing the path that may be taken through a data structure during the execution of a procedure.

DATA ANALYSIS
A disciplined approach to analyzing the meaning and properties of the data elements in existing clerical forms and computer files, independently from the systems which produce and use this data.

DATA DEPENDENCY
The situation where a process creates or modifies some data, which is subsequently used by some other process.

DATA FLOW
A requirement for a data view to pass between two designed elements, each being a business system, procedure, data store or terminator.

DATA FLOW DIAGRAM
A diagram which shows the data object types output by one activity and subsequently used by another activity.

DATA MANAGEMENT SYSTEM
A software product which takes care of files or databases and through which modules retrieve and update the data these files contain.

DATA MODEL
A diagram showing a consistent structure of related information objects of one or more types.

DATA OBJECT
An object which forms part or all of a data structure.

DATA SHARING
The situation where one type of data is used to support more than one business activity.

DATA STORAGE STRUCTURE
The way in which data is organized in storage, which determines the performance of the business systems in their use of data, but which is not known to programs.

DATA STORE
A repository of data, possibly temporary, of which users are aware, and from which data may be read repeatedly and non-destructively.

DATA STRUCTURE
A designed and defined collection of record types, linkages, fields, entry points and integrity rules required to support one or more business systems.

DATA VALUE
The content of an occurrence of a data field.

DATA VIEW
An organized collection of fields or layouts which is meaningful to a procedure, business system or organizational unit.

DATABASE
A discrete collection of related records, linkages and control data managed by one data management system.

DATABASE MANAGEMENT SYSTEM
The situation where data, which is structured to model the relationships inherent in the enterprise, is shared between several business systems.

DECISION ANALYSIS
The study of the decisions taken by management and the factors considered in arriving at each decision.

DECISION EVENT
The taking of a decision within the enterprise, which triggers the execution of one or more processes.

DECISION FACTOR
An item of information which is taken into consideration when making a decision.

DECISION TABLE
A diagram which consists of a condition table, together with a list of the processes which are executed for each combination of predicate values in the table.

DECOMPOSITION
The step-by-step breakdown into increasing detail either of functions, eventually into processes and then into actions, or of entity types into subtypes.

DECOMPOSITION DIAGRAM
A structure which shows the breakdown of objects of a given type into progressively increasing detail.

DEFINITION
A description of an element type which is sufficiently rigorous to determine whether or not a given element is of the type defined.

DEPENDENCY
See ACTION DEPENDENCY, PROCESS DEPENDENCY, SYSTEM DEPENDENCY, BUSINESS FUNCTION DEPENDENCY.

DERIVED ATTRIBUTE
An attribute whose values can each be calculated or deduced from the values of other predicates.

DESCRIPTIVE ENTITY TYPE
An entity type whose entities each describe an entity of one other type.

DESIGN AREA
A collection of closely related processes, relationships and attributes in support of which, one or more business systems are designed together.

DESIGN STAGE
The period in the systems life cycle in which a complete and detailed specification is produced of the business systems and data structures needed to support a defined area within the enterprise.

DESIGNED ATTRIBUTE
An attribute which has been invented in order to overcome constraints or to simplify the operation of a system.

DESIGNER
A person who works on the tasks of the design stages.

DEVELOPMENT COORDINATION
The organizational unit responsible for the extension and maintenance of the architectures, for the control of data and for other aspects of systems development which should not be within the scope of one business system.

DISASSOCIATE
An action by which one of the entities in a grouping, is removed from that grouping or by which one of the records in a linkage is removed from that linkage.

DOMAIN
A meaningful collection of values from which each of the values of one or more attributes and/or fields must be taken.

DUPLICATED ACTIVITY
An activity which appears at more than one point in an activity hierarchy.

ELEMENTARY ACTION
That part of a sequence of processing which is undertaken on a single entity of a type without involving any other entities.

ELEMENTARY PROCESS
A process which when complete leaves the system in a self-consistent state.

ENCYCLOPAEDIA
The database which contains the detailed description of the enterprise, its data resources and business systems, as populated progressively during each stage of Information Engineering.

ENTERPRISE
a business or government organization or part of an organization.

ENTITY
A fundamental thing of relevance to the enterprise, about which data could be kept.

ENTITY ANALYSIS
A disciplined approach to understanding and documenting the things of interest to the enterprise, independently from the activities which take place in the enterprise.

ENTITY ANALYSIS OBJECT
An object forming part of the entity relationship model.

ENTITY LIFE
A description of what happens to one entity from the time it becomes of interest to an enterprise to the time it ceases to be interest to an enterprise.

ENTITY RELATIONSHIP DIAGRAM
A diagram representing entity types and the relationships between them, and certain of their important properties.

ENTITY RELATIONSHIP MODEL
A detailed and structured representation of all the results of Entity Analysis.

ENTITY STATE
A definable, discrete period in the life of an entity.

ENTITY SUBTYPE
A collection of entities of the same type but to which a narrower definition and additional common predicates apply.

ENTITY TYPE
The collection of all the entities to which a specific definition and common predicates apply.

ENTITY TYPE DECOMPOSITION DIAGRAM
A structure showing the breakdown of entity types into progressively smaller collections of entities.

ENTITY TYPE INVOLVEMENT MATRIX (ENTITY TYPE/PROCESS MATRIX)
A matrix which shows for entity types, the processes with which they are involved and by which actions.

ENTITY TYPE LIFE CYCLE
A description of the sets of processes and events that can act on an entity in each of the states that are possible in the lives of entities of that type. A description of what happens during the lives of entities of one type.

ESTABLISH
An action by which a new entity or record is established.

ESTABLISHMENT
The intial state in the life of an entity.

EVENT
The availability of an information or data view, or resource, which is produced by an external object or the passing of a specific point in time, which enables executions of one or more activities.

EXCLUSIVE DEPENDENCIES
Two or more dependencies between activities such that each execution of the one process results in or follows from the execution of only one of the other activities.

EXCLUSIVE RELATIONSHIPS
Two or more relationships, under which each entity of a given type may participate in any one, but not more than one grouping.

EXCLUSIVE SUBPROCESSES
Two or more processes of which only one may be executed during each execution of the process of which they form a part.

EXECUTION CONDITION
A rule expressed in terms of predicates and/or constants which states when an activity may or may not be executed.

EXTERNAL OBJECT
An object which is outside of a business area but which interacts with it.

FACT
A definite predicate or group of predicates describing something which is known.

FIELD
A type of container for either data values or headings.

FILE
A container for either records, keys or code statements.

FLOATING ACTIVITY
An activity which has at least one non-specific (ie. implicit) dependency on another activity.

FLOATING CONDITION
A condition (either pre- or post-) which causes a floating activity.

FOREIGN CLASSIFIER
Entities of a specific type are partitioned into subtypes depending on the values of one or more attributes of a related entity type.

FOREIGN IDENTIFIER
Entities of a specific type are identified in whole or in part by one or more attributes of a related entity type.

FULLY OPTIONAL RELATIONSHIP
A relationship under which entities of both entity types can exist without participating in some grouping.

FULLY OPTIONAL DEPENDENCY
A process dependency under which each process may be executed without resulting in or following from the execution of the other.

FUNCTION
See BUSINESS FUNCTION.

FUNCTION ANALYSIS
A disciplined approach to understanding and documenting the detailed activities in the enterprise, independently from its organization structure.

FUNCTION ANALYSIS OBJECT
An object forming part of the function model.

FUNCTION DECOMPOSITION DIAGRAM
A structure which shows the breakdown of functions into progressively increasing detail.

FUNCTION DEPENDENCY
See BUSINESS FUNCTION DEPENDENCY.

FUNCTION DEPENDENCY DIAGRAM
A diagram which shows that each function may depend on other functions.

FUNCTION HIERARCHY
See FUNCTION DECOMPOSITION DIAGRAM.

FUNCTIONAL DEPENDENCY
A dependency between two fields, such that the value of the first determines which layout occurrence contains each value of the second.

FUNDAMENTAL ENTITY TYPE
An entity type whose entities are each not dependent on any other entities for their existence.

GENERIC
A classifying term.

GLOBAL MODEL
A diagram which represents all of the enterprise or that part of the enterprise so far analyzed.

GROUPING (Occurrence)
A collection of entities of one or two types, associated by virtue of a defined relationship between them.

HARDWARE PRODUCT
A type of hardware which is obtainable from a specific supplier.

IDENTIFIER
An attribute whose values assist in enabling each entity of a specific type to be distinguished from others of the same type.

IDENTIFYING ATTRIBUTE
See IDENTIFIER.

IMPLEMENTATION AREA
A collection of closely related procedures and data stores in support of which one, or a part of one, business system will undergo technical design, construction and transition.

INFORMATION ARCHITECTURE
A global structure expressed in terms of an entity relationship model and a function dependency model, based upon which individual business sytems can be developed, in the knowledge that these may be readily integrated and share data at some future time.

INFORMATION ENGINEERING
A methodology for developing integrated business systems based on the sharing of common data and procedures.

INFORMATION ENGINEERING FACILITY
A software product which includes an analyst/designer's workbench and a central encyclopaedia, and which supports all stages of Information Engineering from Information Strategy Planning to Transition.

INFORMATION NEED
An unstrucutred statement describing a type of information required by an organizational unit to enable it to meet its objectives and support its functions.

INFORMATION OBJECT
An object which is analyzed during Entity Analysis or designed during Data Design.

INFORMATION STRATEGY PLANNING
The period in the systems life cycle in which an information architecture, a business systems architecture and a technical architecture are first produced and under which a consistent series of business systems will be developed.

INFORMATION SYSTEM (Object Subtype)
A business system to support the decision-making processes within a function, providing accesses to an organized consistent collection of data, in ways which are not pre-determined.
OR A means of recording and communicating information to satisfy the requirements of all users and the processes they carry out.

INFORMATION VIEW
A collection of associated predicates input to or output from a process or business function, on which a process or business function is dependent.

INFRASTRUCTURE
The implemented form of that part of an architecture which is sharable within the enterprise and provides a common service at corporate or local levels (note SUPERSTRUCTURE).

INTEGRITY CONDITION
A rule expressed in terms of predicates and/or constants which states a constraint inherent to the area under analysis.

INTERACTION ANALYSIS
The study of the associations between entity analysis objects and function analysis objects.

INVOLUTED DEPENDENCY
The situation where the execution of an activity may lead to a further execution of the same activity (a k a RECURSION).

INVOLUTED RELATIONSHIP
A relationship in which the two entities of every pairing are from the same entity type.

ISOLATED ENTITY TYPE
An entity type which does not participate in any relationship.

LIFE CYCLE ANALYSIS
The analysis of what can happen during the lives of entities of one type.

LIFE CYCLE DIAGRAM (aka STATE TRANSITION DIAGRAM)
A diagram showing all the possible states in the lives of the entities of one type and the processes which cause changes in their states.

LIFE CYCLE MATRIX (aka STATE TRANSITION MATRIX)
A matrix showing for each state, applicable to the entities of one type, the processes which are valid and those which cause a change in state.

LINKAGE
A connection between two related records by which records may be accessed, but of which modules need have no knowledge.

LOCATION
A physical or hypothetical place of interest to the enterprise because either some of its activities are carried out there or information must be communicated to it.

LOCATION TYPE
A general category which is used to express the nature of locations.

MAJOR ENTITY TYPE
An entity type which is considered fundamental to an enterprise and which is used by a number of parts of that enterprise.

MANDATORY DEPENDENCY
A dependency under which every execution of the first activity is followed by an execution of the second activity.

MANDATORY MEMBERSHIP
Membership of an entity type in a relationship such that each entity of the type must participate in a pairing under that relationship.

MANDATORY PARTITIONING
A partitioning for which every entity of its type must be a member of some subtype.

MANDATORY RELATIONSHIP
A relationship under which all entities of both the entity types involved, must participate in some grouping.

MANDATORY SUBPROCESS
A process which is executed for every execution of the process of which it forms a part.

MANY-TO-MANY RELATIONSHIP
A business reason under which one or more entities in one role may be associated with one or more entities in another role.

MAPPING
A technique whereby a view taken for one purpose is converted to or reconciled with a view taken for a different purpose by following defined rules or guidelines.

MEMBER
An entity which participates in some defined collection of entities, which may be an entity type, entity subtype or grouping.

META OBJECT (Generic)
an object type, association type or property type.

MISSION (Object Type)
A general statement of the purpose and nature of the enterprise.

MODEL
A representation of specific aspects of part or all of an enterprise.

MODEL VERSION
A version of a model, for an Information Engineering stage, which is specific to a project and may therefore be one of several in existence at any time. Model versions may overlap in that object versions may appear in the definition of more than one model version.

MODIFY ATTRIBUTE
An action by which a value is replaced by a new value during the execution of a process.

MODIFY ENTITY
An action by which one or more attribute values of an entity are changed during the execution of a process.

MODULE
A sequence of statements written in a specific language which is stored as a single unit and which is used in support of steps.

MORPHOLOGY
The shape of a diagram.

MULTI-VALUED ATTRIBUTE
An attribute where more than one value can describe an entity at any given time.

NORMALIZATION
The decomposition of a data structure, to remove any implicit functional dependencies between objects within the structure.

OBJECT
A meta-entity which is represented by Information Engineering

OBJECT VERSION
An object appearing in a model version.

OBJECTIVE
A general statement about a direction in which the enterprise intends to go, to help further its mission.

ONE-TO-MANY RELATIONSHIP
A business reason under which an entity in one role may be associated with one or more entities in another role.

ONE-TO-ONE RELATIONSHIP
A business reason under which an entity in one role may be associated with one and only one entity in another role.

OPTIONAL MEMBERSHIP
Membership of an entity type in a relationship, such that entities of the type may exist without participating in a pairing under the relationship.

OPTIONAL PARTITIONING
A partitioning for which some entities of the entity type may be members of any of its subtypes.

OPTIONAL SUBPROCESS
A process which may or may not be executed during each execution of the process of which it forms a part.

ORGANIZATIONAL ELEMENT
An aspect of the enterprise's organization of interest to an Information Engineering project.

ORGANIZATIONAL ROLE
A type of position which can be allocated to a person within an organizational unit and which defines the job they will do.

ORGANIZATIONAL UNIT
A named collection of people or of smaller organizational units, used to structure the enterprise, or an external body with which the enterprise deals.

PAIRING
Two entities of one or two types associated by virtue of a defined relationship.

PARALLEL DEPENDENCY
A situation where there is more than one dependency between two processes, any one of which can apply to a given execution of the processes.

PARALLEL RELATIONSHIPS
Two or more relationships are able to associate entities of the same two roles.

PARTITIONING
A basis for subdividing the entities of one type into subtypes.

PARTLY OPTIONAL DEPENDENCY
A process dependency under which one of the two processes cannot be executed unless the other process is also executed.

PARTLY OPTIONAL RELATIONSHIP
A relationship under which entities of one of the related entity types can exist without participating in some grouping.

PERSON
An individual with responsibility for one or more objects or who takes on a defined role within the organization and carries out allocated tasks.

PREDICATE
An attribute or a relationship member.

PREDICATE VALUE
An attribute value or a pairing membership.

PRELIMINARY DATA STRUCTURE
The first formal data structure diagram conforming to the target DBMS, without taking into account performance considerations

PRELIMINARY DATA STRUCTURE DIAGRAM
A representation of the preliminary data structure.

PREMISE
A statement about the state of an enterprise from which another is inferred. The basis for deriving a conclusion.

PRIMARY IDENTIFIER
The preferred means of distinguishing each entity of a type from others of the same type.

PROBLEM ANALYSIS
The study of the difficulties associated with each of the functions or processes of an enterprise or a business area and the limitations of the mechanisms currently supporting them.

PROCEDURE
A method by which one or more elementary processes may be carried out.

PROCEDURE DESIGN
The task of specifying the steps, data input and output, and detailed logic of a procedure.

PROCESS
A defined business activity, executions of which may be identified in terms of the input and/or output of entities of specific types, or of data about entities of specific types.

PROCESS ACTION DIAGRAM
A representation of the logic of a process in terms of the actions carried out on each entity analysis object involved and the conditions constraining these actions.

PROCESS DECOMPOSITION DIAGRAM
A structure which shows the breakdown of processes into progressively increasing detail.

PROCESS DEPENDENCY
An association between processes such that an execution of a first process must be or may be followed by an execution of another process. The dependency may be between executions of the same process.

PROCESS DEPENDENCY ANALYSIS
The analysis of the sequences in which processes can be executed and the attributes which are passed from one process to another.

PROCESS DEPENDENCY DIAGRAM
A diagram which shows why for each process, an execution may depend upon the prior execution of other processes.

PROCESS HIERARCHY
See PROCESS DECOMPOSITION DIAGRAM

PROCESS LOGIC ANALYSIS
An analysis of the inherent logic of a process in terms of the entity types, relationships and attributes involved, the conditions constraining the execution of its subprocesses, and the algorithms used.

PROCESS LOGIC DIAGRAM
A diagram showing the inherent logic of a process, in terms of the sequence in which entity types and relationships are involved.

PRODUCTION
The period in the systems life cycle in which computer applications provide support to the areas of the enterprise for which they were designed.

PROGRAM
A sequence of instructions to a computing device which supports one or more procedures and which may be executed independently of other such sequences.

PROJECT
An activity established to carry out one or more consecutive stages in the life cycle of a system.

PROPERTY
An attribute value which describes an object.

PROTOTYPE
The original or model on which something is formed. An example displaying characteristic of a class. Something exhibiting features of another thing developed in a later age. (Webster's)

PSEUDO CODE
See STRUCTURED LANGUAGE.

RECORD
A collection of occurrences of fields which is read or written as a single unit, during the execution of a module.

RECORD LAYOUT
A collection of related fields, which represent data in a structure visible to both modules and the data management system.

RECORD TYPE
See RECORD LAYOUT.

RECURSION
The situation where the execution of an activity may lead to a further execution of the same activity (see INVOLUTED DEPENDENCY)

REDUNDANT DEPENDENCY
A process dependency which exists only because each of its processes have a dependency, directly or indirectly, with some third process.

REDUNDANT RELATIONSHIP
A relationship where each of its pairings can be derived from pairings under other, more basic, relationships.

RELATIONSHIP
A reason of relevance to the enterprise why entities from one or from two entity types may be associated.

RELATIONSHIP MEMBERSHIP (Object Type)
The participation of an entity type in a relationship.

RELATIONSHIP ROLE
A business reason whereby entities of a specific type may participate in groupings under a relationship, either as plural members or as single members.

RELATIONSHIPS OF FIXED CARDINALITY
A relationship under which each grouping has the same number of entities participating from the one or two entity types involved.

RESOURCE
A material requirement of the enterprise.

RESOURCE HANDLING PROCESS (Property)
A process concerned with the planning, acquisition, use or disposal of resources of the enterprise.

ROOT FUNCTION
A function which is not itself a subfunction of any other function.

RULE
A statement involving predicates, actions and conclusions which reflects some piece of knowledge.

SCOPE
A defined subset of objects which are the subject of a specific project.

SELECTION CONDITION (Property)
A rule expressed in terms of predicates and/or constants, used to select one or more entities of a given type for involvement during the execution of a process.

SELECT (Property)
An action by which a value of an attribute is used to select, or to assist in selecting, an entity for involvement in the execution of a process, or by which a data value is used to select, or to assist in selecting, a record for use during the execution of a module.

SIMILARITY COEFFICIENT
The sum of all weighted similarities of a pair of object types in respect of their association with each other or with a third object.

SOFTWARE PRODUCT
A documented and complete collection of statements which is obtained from a specific supplier and which may be executed directly or indirectly on a computing device.

SOLITARY ATTRIBUTE
An attribute which is the only one for its entity type.

SOLITARY ENTITY
An entity which is the only one of its type.

STABILITY ANALYSIS (Task)
The study of the impact that a sample of potential changes in the enterprise would have on the analysis area, as represented by the models and supporting documentation.

STAGE
A structured set of Information Engineering tasks, the end result of which is a major deliverable and a decision point in the systems life cycle.

STATEMENT
A unit of a computer language which represents either an instruction, or a definition of a data structure, or a record in a table.

STEP
A discrete activity recognised by a user or operator and carried out as part of a procedure.

STRATEGIC DECISION
A decision concerned with the objectives of an enterprise and with planning the resources that are necessary to meet these objectives.

STRATEGIC PROCESS
A process concerned with establishing the objectives of an enterprise and ensuring that it has resources to achieve them.

STRATEGY
A means by which the enterprise will deploy its resources to achieve an objective.

STRATEGY STAGE
See INFORMATION STRATEGY PLANNING

STRUCTURED LANGUAGE
Statements written in a subset of a natural language within a disciplined organization, such that they can readily be translated into a language that can be processed by an intelligent device.

STRUCTURED WALK-THROUGH
A symbol-by-symbol verbal explanation of a diagram by the analyst or designer responsible, with the objective of eliminating errors and inconsistencies.

SUBFUNCTION
A meaningful group of business activities and decisions within a function.

SUBJECT AREA
A natural subdivision of an enterprise centered on a major resource or product or activity of the enterprise.

SUBJECT AREA ASSOCIATION(/RELATIONSHIP)
A reason of relevance to the enterprise why subject areas may be associated.

SUBJECT ENTITY TYPE
The resource or activity upon which a subject area is centered.

SUBPROCESS
A specific activity which is executed as part of the execution of a process.

SUBSYSTEM
A subdivision of a business system into a small collection of procedures for development planning, education or control purposes.

SUPERSTRUCTURE
The implemented form of that part of an architecture consisting of objects each of which provides a specific service to a particular part of the enterprise. There are two categories of superstructure object - one dependent on infrastructure support, the other independent.

SUPPORT
The type of support given by a current or planned business system to a data or activity object.

SYMBOL
A shape used in a diagram to denote a specific object type or a property type.

SYNONYM
An alternative name of an object.

SYSTEM
A set or complex of inter-related and interacting elements.

SYSTEM ACTIVITY
An object which forms part or all of a system structure.

SYSTEM DEPENDENCY
An association between two systems which exists because information originating in one is required by the other.

SYSTEM DEVELOPMENT APPROACH
A general direction for doing something. In system development an approach provides a general framework within which development is carried out, and this framework is based on fundamental beliefs. These beliefs may be axiomatic in that they do not necessary have to be proven. A hierarchy of system development approaches can be constructed, based on the orientation of a particular set of approaches.

SYSTEM DEVELOPMENT METHOD
An orderly arrangment of ideas that aids a particular activity (such as system design or system analysis). A method usually contains an inherent logical assumption and it is based on a theoretical concept. Thus, a system development method is used to practise a system development approach. (Indeed, a system development approach cannot be practised without a system development method.) Some system development methods can be used within more than one system development approach.

SYSTEM DEVELOPMENT METHODOLOGY
A collection of interconnecting methods and techniques, normally within the framework of an approach. A methodology represents a packaging of practical ideas and proven practises for a given area of activity. As an example, within the structured approach, programming methodologies and system development methodologies have been developed.

SYSTEM DEVELOPMENT TECHNIQUE
A predominantly mechanical way of doing something. System development techniques therefore provide the detailed guidelines for using a system development method, and a system development technique will often require that a specific tool be used. For example, those documentation methods that are based on the assumption that system design should be represented in a pictorial and a diagrammatic format, require that both techniques and tools be used in order to draw the diagrams.

SYSTEM OBJECT
An object which forms part of the Business Systems Architecture.

SYSTEMS LIFE CYCLE
The stages and tasks in the development and productive use of a system from its inception to its demise.

TACTICAL DECISION
A decision concerning changes in the allocation of resources or in the ways in which an enterprise operates.

TACTICAL PROCESS
A process concerned with the allocation and efficient utilization of the resources of an enterprise.

TASK
A defined unit of work for one or more persons within a project.

TECHNICAL ARCHITECTURE
A structure which summarizes the mixture of hardware, software and communication facilities which supports or will support the business systems of the enterprise.

TECHNICAL DESIGN
That part of the Design Stage during which the system is refined to achieve the most economic and efficient performance using the chosen technology.

TERMINAL
A hardware product which displays data interactively to the user.

TERMINATION STATE
A final state in the life of an entity.

TIME DEPENDENCE
The situation where the cardinality or optionality of a relationship, can differ depending on whether or not its meaning takes account of the passage of time.

TIME EVENT
The passage of a specific time period which triggers the execution of one or more processes.

TIMELINESS
The time delay involved or expected in the up-dating of objects of a specific type of data object type.

TRANSACTION
A complete execution of a procedure.

TRANSFER
An action by which one of the entities in a grouping is transferred to a different grouping under the same relationship, or by which one of the records in a linkage is transferred to a different linkage.

TRANSITION
The period in the system life cycle in which the new business systems to support a defined area within the enterprise gradually replace or are interfaced to the existing systems.

TRANSITIVE DEPENDENCY
A dependency between two elements, which is due to the first element being dependent on some other element, which in turn is dependent on the second.

USER
A person or organizational unit of the enterprise, responsible for applying an automated procedure to support the executions of a process.

VALUE
See ATTRIBUTE VALUE and DATA VALUE

VERSION
A view of an entity which ceases to be the current view, if any one of the predicates of that entity are changed in value.

WEIGHTING FACTOR
A factor to be applied to a specific pair of values in order to derive a component of a similarity coefficient.

END.

APPENDIX X5 LIST OF PUBLICATIONS

Much of this thesis has been published elsewhere during the lifetime of the research. The thesis has taken many of these publications and expanded them to be complete. In chronological order of publication, the publications relating to this thesis are:

FELDMAN, P	[FELD,83]
FELDMAN, P & MILLER, D	[FEMI,85]
FELDMAN, P & FITZGERALD, G	[FEFI,85a]
FELDMAN, P & FITZGERALD, G	[FEFI,85B]
FELDMAN, P & MILLER, D	[FEMI,86]
FELDMAN, P, MACDONALD, I & MABEY, C	[FMM,86] (still to be published)

APPENDIX X6 - DIAGRAMMER LISTINGS

This appendix lists the programs of the diagrammer described in chapter E.

There are two programs, both PASCAL, PLOT and DRAW.

PLOT calcluates the ideal positions of the objects on a diagram.

DRAW displays and maniplates the objects as required.

```
PROGRAM PLOT(infile, pltfil, Input , Output);
(* PROGRAM TO DISCOVER GOOD POSITIONS TO PUT ENTITES & RELATIONSHIPS *)
     CONST MAXNAMELTH = 32;
           NILSTRING = '                                        ';
           end_of_entity_marker = '$$$$$                              _    ';
           XGAP = 8;
           HALFXGAP = 4;
           YGAP = 4;
           HALFYGAP = 2;
           GAPBETRELS = 1;
           WIDTHOFENTITYBOX = 8;
           HEIGHTOFENTITYBOX = 4;
           OOFTRIES = 4;
           PROPORTION = 2;
           infinite_grad = Maxint;
           NULLCOORD = NIL;
           NOINTERSECTION = NIL;
           MAXNOXRELS = 8; (* MAX. NO OF RELS WHICH CAN START FROM THE
                              X EDGE OF AN ENTITY BOX *)
           MAXNOYRELS = 4;
           GTXBND = 2;
           LESSXBND = 1;
           GTYBND = 1;
           LESSYBND = 4;
           FREE = NIL;
           INTRELPENALTY = 5;


     TYPE RELATONPTR = ↑ RELTYP;
          ENTITYRELPTR = ↑ ENTRELLINK;
          ENTITYPTR = ↑ ENTTYP;
          SUBENTITYPTR = ↑ SUBENTTYP;

          NAME = STRING( MAXNAMELTH );
          filname = String( 8 );
          OPTTYP = 'M'..'O';
          DEGREETYP =  'M'..'O';
          TYPOFRELATON =  'A'..'P';

          (* DECLARATIONS OF RECORDS FOR FILES *)

          ENTTYP = RECORD
               ENTITYNAME: NAME;
               ent_name_lth: Integer;
               RELATONSPTR: ENTITYRELPTR;
               SUBENTS: SUBENTITYPTR;
               NEXTENTITYPTR: ENTITYPTR
          END (* RECORD *);

          SUBENTTYP = RECORD
               ENTPTR: ENTITYPTR;
               SUBENTPTR: ENTITYPTR;
               NEXTSUBENT: SUBENTITYPTR
          END (* RECORD *);
```

```
entrellink = Record
     entpt: entityptr;
     RELPT: RELATONPTR;
     OPTALITY: OPTTYP;
     DEGREE: DEGREETYP;
     RELATONTYP: TYPOFRELATON;
     NEXTRELFORENTITY: ENTITYRELPTR
END (* RECORD *);

RELTYP = RECORD
     ACTIVEPTR: ENTITYRELPTR;
     ACTIVERELNAME: NAME;
     act_rel_name_lth: Integer;
     PASSIVEPTR: ENTITYRELPTR;
     PASSIVERELNAME: NAME;
     pas_rel_name_lth: Integer;
     NEXTRELPTR: RELATONPTR
END (* RECORD *);


COMPONENT = INTEGER;

RECTEDGE = (NILEDGE , TOP , RIGHT , BOTTOM , LEFT );
LINEDIRECTION = ( DIRT , HOR , VERT );


RECTCOORD = ARRAY[1..4] OF COMPONENT;

ENTPLOTPTR = ↑ENTPLOTTYP;
RELPLOTPTR = ↑RELPLOTTYP;

COORDPTR = ↑COORDINATE;
BOXPTR = ↑BOXTYP;
COORDLISTPTR = ↑COORDLST;

EDGESTARTS = ARRAY[1..MAXNOXRELS] OF COORDLISTPTR;
EDGERELSTARTS = ARRAY[TOP..LEFT] OF EDGESTARTS;

ENTPLOTTYP = RECORD
     ENTITY: ENTITYPTR;
     NOOFRELS: INTEGER;
     BOXPLOT: BOXPTR;
     NEXTENTRY: ENTPLOTPTR
END (* RECORD *);

RELPLOTTYP = RECORD
     RELATON: RELATONPTR;
     CRDLIST: COORDLISTPTR;
     NEXTENTRY: RELPLOTPTR
END (* RECORD *);

COORDLST = RECORD
     STARTCOORD: COORDPTR;
     ENDCOORD: COORDPTR;
     LINEGRAD: REAL;
```

```
            LINECONST: REAL;
            STRORFIN: BOOLEAN;
            STRTEDGE: RECTEDGE;
            ENDEDGE: RECTEDGE;
            PREVCOORD: COORDLISTPTR;
            NEXTCOORD: COORDLISTPTR
        END (* RECORD *);

        COORDINATE = RECORD
            X: COMPONENT;
            Y: COMPONENT
        END (* RECORD *);

        BOXTYP = RECORD
            CENTRE: COORDPTR;
            XCOORDS: RECTCOORD;
            YCOORDS: RECTCOORD;
            RELSONEDGE: EDGERELSTARTS
        END (* RECORD *);



    VAR
        (* POINTERS TO THE DATA LISTS HOLDING ALL INFORMATION *)
        ENTITYLIST: ENTITYPTR;
        TAILENTITYLIST: ENTITYPTR;
        RELATONLIST: RELATONPTR;
        TAILRELATONLIST: RELATONPTR;

        INFILNAME, plot_fil_name: filname;

      HDENTPLOT,
      TLENTPLOT,
      HDSORTEDENT: ENTPLOTPTR;

      HDRELPLOT,
      TLRELPLOT: RELPLOTPTR;
      HDRELUNPLOT,
      TLRELUNPLOT: RELPLOTPTR;

      infile, PLTFIL: TEXT;

      ORIGN: COORDPTR;
      strtpt: coordptr;
      INFINITY: COORDPTR;

      OUT: TEXT;
      LASTBOXBELOW: BOOLEAN;
      sorted, ascending, place_by_entity: Boolean;
      average_rel: Boolean;

    FUNCTION ADDNEWENTITYENTRY: ENTITYPTR ;
    (* ADD A NEW ENTRY TO THE TAIL OF THE ENTITY LIST *)
    BEGIN
        IF ENTITYLIST = NIL THEN
```

```
        BEGIN
             (* FIRST ENTRY IN LIST *)
             NEW( ENTITYLIST );
             TAILENTITYLIST := ENTITYLIST (* HD. & TL. ARE THE SAME *)
        END
        ELSE BEGIN
             (* ADD AN ENTRY TO TAIL OF LIST *)
             NEW( TAILENTITYLIST↑.NEXTENTITYPTR );
             TAILENTITYLIST := TAILENTITYLIST↑.NEXTENTITYPTR
        END (* ELSE *);
        (* SET NEW ENTRY DEFAULT VALUES *)
        WITH TAILENTITYLIST↑ DO
        BEGIN
             ENTITYNAME := nilstring;
             RELATONSPTR := NIL;
             SUBENTS := NIL;
             NEXTENTITYPTR := NIL
        END (* WITH *);
        ADDNEWENTITYENTRY := TAILENTITYLIST
    END (* ADDNEWENTITYENTRY *);

    FUNCTION ADDNEWRELATIONENTRY: RELATONPTR ;
    (* ADD A NEW ENTRY TO THE TAIL OF THE RELATON LIST *)
    BEGIN
        IF RELATONLIST = NIL THEN
        BEGIN
             (* FIRST ENTRY IN LIST *)
             NEW( RELATONLIST );
             TAILRELATONLIST := RELATONLIST (* HD. & TL. ARE THE SAME *)
        END
        ELSE BEGIN
             (* ADD AN ENTRY TO TAIL OF LIST *)
             NEW( TAILRELATONLIST↑.NEXTRELPTR );
             TAILRELATONLIST := TAILRELATONLIST .NEXTRELPTR
        END (* ELSE *);
        (* SET NEW ENTRY DEFAULT VALUES *)
        WITH TAILRELATONLIST↑ DO
        BEGIN
             ACTIVERELNAME := nilstring;
             ACTIVEPTR := NIL;
             PASSIVEPTR := NIL;
             PASSIVERELNAME := nilstring;
             NEXTRELPTR := NIL
        END (* WITH *);
        ADDNEWRELATIONENTRY := TAILRELATONLIST
    END (* ADDNEWRELATIONENTRY *);

    FUNCTION FINDENTITY( ENTNAME: NAME): ENTITYPTR ;
    (* FIND AN ENTITY IN THE ENTITY LIST AND RETURN A POINTER TO IT *)
    VAR ENTPT: ENTITYPTR;
        NOTFOUND: BOOLEAN;
    BEGIN
        NOTFOUND := TRUE; (* HAVE TO USE, CAUSE PASCAL CHECKS ALL PARTS
                             OF A CONDITION *)
        ENTPT := ENTITYLIST;
```

```
     (* SEARCH LIST UNTIL ENTITY FOUND OR END OF LIST *)
     WHILE (ENTPT <> NIL) AND NOTFOUND DO
          (* SEE IF HAVE ENTITY DESIRED *)
          IF ENTNAME  = ENTPT↑.ENTITYNAME THEN
               NOTFOUND := FALSE (* ENTITY FOUND *)
          ELSE
               (* FIND NEXT ENTITY TO CHECK THAT ONE *)
               ENTPT := ENTPT^.NEXTENTITYPTR;
     (* RESULT IS NIL IF NO SUCH ENTITY *)
     FINDENTITY := ENTPT
END (* FINDENTITY *);

FUNCTION ADDENTITYRELLINK( ENPT: ENTITYPTR ): ENTITYRELPTR ;
(* CREATE A NEW ENTITY/REL LINK AND ADD IT TO THE LIST FOR AN ENTITY *)
VAR ERPT: ENTITYRELPTR;
    TEMPERPT: ENTITYRELPTR;
BEGIN
     (* SET UP NEW ENTRY *)
     NEW( ERPT );
     IF ENPT↑.RELATONSPTR = NIL THEN
          (* FIRST ONE FOR THE ENTITY *)
          ENPT↑.RELATONSPTR := ERPT
     ELSE BEGIN
          (* FIND ENTRY IN LIST AND ADD NEW ENTRY THERE *)
          TEMPERPT := ENPT .RELATONSPTR;
          WHILE TEMPERPT .NEXTRELFORENTITY <> NIL DO
               TEMPERPT := TEMPERPT↑.NEXTRELFORENTITY;
          (* ADD NEW ENTRY *)
          TEMPERPT↑.NEXTRELFORENTITY := ERPT
     END (* ELSE *);
     (* SET UP DEFAULT VALUES *)
     WITH ERPT↑ DO
     BEGIN
          ENTPT := ENPT;
          RELPT := NIL;
          OPTALITY := 'N';
          DEGREE := 'N';
          RELATONTYP  := 'N';
          NEXTRELFORENTITY := NIL
     END (* WITH *);
     ADDENTITYRELLINK := ERPT
END (* ADDENTITYRELLINK *);

PROCEDURE ADDSUBENTITY( SUBENTPT, SUPERENTPT:ENTITYPTR);
(* ADD A NEW SUB-ENTITY ENTRY TO THE SUPERENT. AND CONNECT PTRS *)
VAR SEPT: SUBENTITYPTR;
BEGIN
     (* FIND END OF SUB-ENTITY LIST OR A DUPLICATE ENTRY *)
     SEPT := SUPERENTPT↑.SUBENTS;
     IF SEPT = NIL THEN
     BEGIN
          (* FIRST ENTRY FOR AN ENTITY *)
          NEW( SUPERENTPT↑.SUBENTS );
          SEPT := SUPERENTPT↑.SUBENTS
     END
```

```
        ELSE BEGIN
             (* SEARCH SUB-ENTITY LIST *)
             WHILE (SEPT^.NEXTSUBENT <> NIL) AND
                   (SEPT^.SUBENTPTR <> SUBENTPT) DO
                  SEPT := SEPT^.NEXTSUBENT;
             (* SEE IF DUPLICATE *)
             IF SEPT^.SUBENTPTR = SUBENTPT THEN
             BEGIN
                  WRITELN( 'DUPLICATE SUB-ENTITY FOUND, ENTITY - ',
                           SUPERENTPT^.ENTITYNAME, ' AND SUB-ENTITY - ',
                           SUBENTPT^.ENTITYNAME );
                  SEPT := NIL (* TO DENOTE INVALID ENTRY *)
             END
             ELSE BEGIN
                  (* CREATE SUB-ENTITY ENTRY AND CONNECT POINTERS *)
                  NEW( SEPT^.NEXTSUBENT );
                  SEPT := SEPT^.NEXTSUBENT
             END (* ELSE *)
        END (* ELSE *);
        IF SEPT <> NIL THEN
        WITH SEPT^ DO
        BEGIN
             (* ADD DEFAULT VALUES TO NEW ENTRY *)
             ENTPTR := SUPERENTPT;
             SUBENTPTR := SUBENTPT;
             NEXTSUBENT := NIL
        END (* THEN WITH *)
   END (* ADDSUBENTITY *);


   PROCEDURE WRITERELNAME( ERPT: ENTITYRELPTR );
   (* TO WRITE THE NAME OF THE SPECIFIED RELATIONSHIP *)
   BEGIN
        WITH ERPT^ DO
        BEGIN
             WRITE(ENTPT^.ENTITYNAME,' ');
             IF RELATONTYP = 'A' THEN
                  WRITE(RELPT^.ACTIVERELNAME, ' ',
                          RELPT^.PASSIVEPTR^.ENTPT^.ENTITYNAME,' ' )
             ELSE
                  WRITE(RELPT^.PASSIVERELNAME, ' ',
                          RELPT^.ACTIVEPTR^.ENTPT^.ENTITYNAME,' ' )
        END (* WITH *)
   END (* WRITERELNAME *);

PROCEDURE READSTUFF;

   Procedure get_next_name( Var nam: name; Var namlth: Integer );
   Var i: Integer;
       ch: Char;
   Begin
        i:= 0;
        nam := nilstring;
        If Not Eof(infile) Then Read(infile , ch);
        While (i < maxnamelth) And (Not Eof(infile))And(Not Eoln(infile)) Do
```

```
        Begin
             i := i + 1;
             nam[i] := ch;
             Read(infile , ch);
        End {While};
        If i<>0 Then
        Begin
             i := i +1;
             nam[i] := ch
        End;
        If Not Eof(infile) Then Readln(infile);
        namlth := i;
        For i := i+1 To maxnamelth Do nam[i] := ' ';
   End {get_next_name};

   PROCEDURE READENTITIES;
   VAR NAM: NAME;
       namlth: Integer;
       ENTPT: ENTITYPTR;
   BEGIN
        get_next_name(nam , namlth);
        While (nam <> end_of_entity_marker) And
              NOT( EOF(INFILE ) ) Do
        BEGIN
             ENTPT := ADDNEWENTITYENTRY;
             ENTPT^.ENTITYNAME := NAM;
             Entpt^.ent_name_lth := namlth;
             get_next_name(nam , namlth)
        END (* WHILE *)
   END (* READENTITIES *);

   FUNCTION READENTRELLINK( RLPT: RELATONPTR ): ENTITYRELPTR;
   VAR ERPT: ENTITYRELPTR;
       ENTPT: ENTITYPTR;
       NAM: NAME;
       namlth, i: Integer;
       CH: Char;
   BEGIN
        get_next_name(nam , namlth);
        (* ENSURE SOMETHING THERE ( MAY BE NO LINK ) *)
        IF (NAM = nilstring) Or Eof(Infile) THEN ERPT := NIL
        ELSE BEGIN
             ENTPT := FINDENTITY( NAM );
             IF ENTPT = NIL THEN
             BEGIN
                  WRITELN( 'MISSING ENTITY - ' , NAM );
                  ERPT := NIL;
                  For i:= 1 to 3 Do Read(infile,ch);
                  If Not Eof(infile) Then Readln(infile );
             END
             ELSE BEGIN
                  ERPT := ADDENTITYRELLINK( ENTPT );
                  ERPT^.RELPT := RLPT;
                  WITH ERPT^ DO
                       READLN( INFILE , OPTALITY , DEGREE , RELATONTYP )
```

```
            END (* ELSE *)
        END (* ELSE *);
        READENTRELLINK := ERPT
    END (* READENTRELLINK *);

    PROCEDURE READRELATIONSHIP;
    VAR RELPT: RELATONPTR;
        NAM: NAME;
        namlth: Integer;
    BEGIN
        Repeat
            get_next_name(nam , namlth);
            If Not Eof(infile) Then
            Begin
                RELPT := ADDNEWRELATIONENTRY;
                WITH RELPT↑ DO
                BEGIN
                    ACTIVERELNAME := NAM;
                    act_rel_name_lth := namlth;
                    ACTIVEPTR := READENTRELLINK( RELPT );
                    get_next_name( passiverelname , pas_rel_name_lth);
                    PASSIVEPTR := READENTRELLINK( RELPT )
                END (* WITH *);
            End
        Until Eof(Infile)
    END (* READRELATIONSHIP *);

BEGIN
    WRITELN('PLEASE WAIT WHILE DATA IS READ IN');
    RESET( INFILE );
    READENTITIES;
    READRELATIONSHIP;
    CLOSE( INFILE )

END (* PROCEDURE READSTUFF *);


    FUNCTION FINDENTPLOTENTRY( ENTPT: ENTITYPTR ): ENTPLOTPTR;
    (* TO FIND THE PLOTTED ENTITY ENTRY OF ENTPT *)
    VAR EPPT: ENTPLOTPTR;
        ENTFND: BOOLEAN;
    BEGIN
        (* SEARCH LIST FOR ENT., IF NIL RELSULT THEN NOT YET PLOTTED *)
        EPPT := HDENTPLOT;
        ENTFND := FALSE;
        WHILE (EPPT <> NIL) AND (NOT ENTFND) DO
            IF (EPPT↑.ENTITY = ENTPT) THEN ENTFND := TRUE
            ELSE
                EPPT := EPPT↑.NEXTENTRY;
        FINDENTPLOTENTRY := EPPT
    END (* FINDENTPLOTENTRY *);

    FUNCTION FINDRELPLOTENTRY( RELPT: RELATONPTR ): RELPLOTPTR ;
    (* TO FIND THE PLOTTED REL. ENTRY OF RELPT *)
    VAR RPPT: RELPLOTPTR;
```

```
    RELFND: BOOLEAN;
BEGIN
     (* SEARCH LIST FOR REL., IF NIL RELSULT THEN NOT YET PLOTTED *)
     RPPT := HDRELPLOT;
     RELFND := FALSE;
     WHILE (RPPT<> NIL) AND (NOT RELFND) DO
          IF (RPPT^.RELATON = RELPT) THEN RELFND := TRUE
          ELSE
               RPPT := RPPT^.NEXTENTRY;
     FINDRELPLOTENTRY := RPPT
END (* FINDRELPLOTENTRY *);

FUNCTION FINDRELUNPLOTENTRY( RELPT: RELATONPTR ): RELPLOTPTR ;
(* TO FIND THE UNPLOTTED REL. ENTRY OF RELPT *)
VAR RPPT: RELPLOTPTR;
    RELFND: BOOLEAN;
BEGIN
     (* SEARCH LIST FOR REL., IF NIL RELSULT THEN NOT YET PLOTTED *)
     RPPT := HDRELUNPLOT;
     RELFND := FALSE;
     WHILE (RPPT   NIL) AND (NOT RELFND) DO
          IF (RPPT .RELATON = RELPT) THEN RELFND := TRUE
          ELSE
               RPPT := RPPT^.NEXTENTRY;
     FINDRELUNPLOTENTRY := RPPT
END (* FINDRELUNPLOTENTRY *);

FUNCTION FINDSORTEDENTRY( ENTPT: ENTITYPTR ): ENTPLOTPTR ;
(* TO FIND THE SORTED ENT. ENTRY OF ENTPT *)
VAR EPPT: ENTPLOTPTR;
    ENTFND: BOOLEAN;
BEGIN
     (* SEARCH LIST FOR ENT., IF NIL RELSULT THEN NOT YET PLOTTED *)
     EPPT := HDSORTEDENT;
     ENTFND := FALSE;
     WHILE (EPPT  NIL) AND (NOT ENTFND) DO
          IF (EPPT^.ENTITY = ENTPT) THEN ENTFND := TRUE
          ELSE
               EPPT := EPPT .NEXTENTRY;
     FINDSORTEDENTRY := EPPT
END (* FINDSORTEDENTRY *);

FUNCTION FINDOTHERENT( ERPT: ENTITYRELPTR ): ENTITYPTR ;
(* TO FIND A POINTER TO THE OTHER ENTITY OF A RELATIONSHIP *)
BEGIN
     IF ERPT..RELATONTYP = 'A' THEN
          (* FIND PASSIVE ENTITY *)
          FINDOTHERENT := ERPT^.RELPT^.PASSIVEPTR^.ENTPT
     ELSE
          FINDOTHERENT := ERPT^.RELPT^.ACTIVEPTR^.ENTPT
END (* FINDOTHERENT *);

PROCEDURE TRANSFRENTTOPLOTTEDLIST( EPPT: ENTPLOTPTR );
(* FIND ENTRY IN UNPLOTTED ENTITY LIST & TRANSFER TO PLOTTED ENTITY LST *)
VAR PREVEPPT: ENTPLOTPTR;
```

```
    ENTFND: BOOLEAN;
BEGIN
     (* FIRST BIND PREV. ENTRY IN UNPLOTTED LIST *)
     PREVEPPT := HDSORTEDENT;
     ENTFND := FALSE;
     (* CHECK THAT NOT AT HEAD OF LIST *)
     IF PREVEPPT <> EPPT THEN
     WHILE (PREVEPPT <> NIL) AND (NOT ENTFND) DO
          IF (PREVEPPT^.NEXTENTRY = EPPT) THEN ENTFND := TRUE
          ELSE
               PREVEPPT := PREVEPPT^.NEXTENTRY;
     IF PREVEPPT = NIL THEN
          (* A DISASTEROUS ERROR HAS SOMEHOW OCCURRED *)
          WRITELN(OUT,' ERROR - ENTITY ',
                ' IS NOT TO BE FOUND IN THE UNPLOTTED LIST' )
     ELSE BEGIN
          (* TRANSFER FROM UNPLOTTED TO PLOTTED LIST *)
          (* FIRST REMOVE FROM UNPLOTTED LIST *)
          IF HDSORTEDENT <> EPPT THEN
               PREVEPPT^.NEXTENTRY := EPPT^.NEXTENTRY
          ELSE BEGIN
               (* AMEND START OF LIST *)
               HDSORTEDENT := HDSORTEDENT^.NEXTENTRY;
               PREVEPPT := HDSORTEDENT
          END (* ELSE *);
          (* NO NEED OF EPPT'S NEXT  ENTRY PTR. ANY MORE *)
          EPPT^.NEXTENTRY := NIL;
          (* NOW PLACE AT TAIL OF PLOTTED LIST *)
          IF HDENTPLOT = NIL THEN
          BEGIN
               (* FIRST ENTRY *)
               HDENTPLOT := EPPT;
               TLENTPLOT := EPPT
          END
          ELSE BEGIN
               (* ADD TO TAIL OF LIST *)
               TLENTPLOT^.NEXTENTRY := EPPT;
               TLENTPLOT := EPPT
          END (* ELSE *)
     END (* ELSE *)
END (* TRANSFRENTTOPLOTTEDLIST *);

PROCEDURE TRANSFRRELTOPLOTTEDLIST( rppt: relplotptr );
(* FIND ENTRY IN UNPLOTTED REL LIST & TRANSFER TO PLOTTED ENTITY LST *)
VAR PREVRPPT: RELPLOTPTR;
    RELFND: BOOLEAN;
BEGIN
     (* FIRST FIND PREV. ENTRY IN UNPLOTTED LIST *)
     PREVRPPT := HDRELUNPLOT;
     RELFND := FALSE;
     IF PREVRPPT <> RPPT THEN
     WHILE (PREVRPPT <> NIL) AND (NOT RELFND) DO
          IF (PREVRPPT^.NEXTENTRY = RPPT) THEN RELFND := TRUE
          ELSE
               PREVRPPT := PREVRPPT^.NEXTENTRY;
```

```
        IF PREVRPPT = NIL THEN
             (* A DISASTEROUS ERROR HAS SOMEHOW OCCURRED *)
             WRITELN(OUT,' ERROR - REL ',
                  ' IS NOT TO BE FOUND IN THE UNPLOTTED LIST' )
        ELSE BEGIN
             (* TRANSFER FROM UNPLOTTED TO PLOTTED LIST *)
             (* FIRST REMOVE FROM UNPLOTTED LIST *)
             IF HDRELUNPLOT <> RPPT THEN
                  PREVRPPT↑.NEXTENTRY := RPPT↑.NEXTENTRY
             ELSE BEGIN
                  HDRELUNPLOT := HDRELUNPLOT↑.NEXTENTRY;
                  PREVRPPT := HDRELUNPLOT
             END (* ELSE *);
             (* NO NEED OF RPPT'S NEXTR ENTRY PTR. ANY MORE *)
             RPPT↑.NEXTENTRY := NIL;
             (* NOW PLACE AT TAIL OF PLOTTED LIST *)
             IF HDRELPLOT = NIL THEN
             BEGIN
                  (* FIRST ENTRY *)
                  HDRELPLOT := RPPT;
                  TLRELPLOT := RPPT
             END
             ELSE BEGIN
                  (* ADD TO TAIL OF LIST *)
                  TLRELPLOT↑.NEXTENTRY := RPPT;
                  TLRELPLOT := RPPT
             END (* ELSE *)
        END (* ELSE *)
   END (* TRANSFRRELTOPLOTTEDLIST *);

   FUNCTION find_average_rel_pos( ENTPT: ENTITYPTR ): coordptr ;
   (* TO FIND THE ENTRY OF AN ENTITY WHICH IS RELATED TO SPECIFIED ONE *)
   VAR ERPT: ENTITYRELPTR;
       EPPT: entplotptr;
       av_coord: coordptr;
       av_x, av_y: component;
       no_rels: Integer;

   BEGIN
        EPPT := NIL;
        av_x := 0; av_y := 0; no_rels := 0;
        IF ENTPT = NIL THEN ERPT := NIL
                       ELSE ERPT := ENTPT↑.RELATONSPTR;
        (* INSPECT EVERY RELATION FOR ENTITY UNTIL FIND ONE WHICH IS
           PLOTTED and then add to averages*)
        WHILE (ERPT<>NIL) DO
        BEGIN
             (* SEE IF NEXT RELATION ENTITY IS PLOTTED *)

             EPPT := FINDENTPLOTENTRY( FINDOTHERENT( ERPT ) );
             If eppt <> Nil Then
             Begin
                  (* related entity is plotted, so add to averages *)
                  av_x := av_x + eppt↑.boxplot↑.centre↑.x;
                  av_y := av_y + eppt↑.boxplot↑.centre↑.y;
```

```
                no_rels := no_rels + 1
            End;
            (* GET NEXT REL. ENTRY *)
            ERPT := ERPT^.NEXTRELFORENTITY
        END (* WHILE *);
        (* set up average coord *)
        If no_rels = 0 Then find_average_rel_pos := nullcoord
        Else Begin
            New( av_coord );
            av_coord^.x := av_x Div no_rels;
            av_coord^.y := av_y Div no_rels;
            find_average_rel_pos := av_coord
        End (* Else *)
    END (* find_average_rel_pos *);

    FUNCTION FINDAPLOTTEDENTITYRELATEDTO( ENTPT: ENTITYPTR ): ENTPLOTPTR ;
    (* TO FIND THE ENTRY OF AN ENTITY WHICH IS RELATED TO SPECIFIED ONE *)
    VAR ERPT: ENTITYRELPTR;
        EPPT: entplotptr;

    BEGIN
        EPPT := NIL;
        IF ENTPT = NIL THEN ERPT := NIL
                       ELSE ERPT := ENTPT^.RELATONSPTR;
        (* INSPECT EVERY RELATION FOR ENTITY UNTIL FIND ONE WHICH IS
           PLOTTED *)
        WHILE (ERPT <> NIL) AND (EPPT = NIL) DO
        BEGIN
            (* SEE IF NEXT RELATION ENTITY IS PLOTTED *)

            EPPT := FINDENTPLOTENTRY( FINDOTHERENT( ERPT ) );
            (* GET NEXT REL. ENTRY *)
            ERPT := ERPT^.NEXTRELFORENTITY
        END (* WHILE *);
        FINDAPLOTTEDENTITYRELATEDTO := EPPT
    END (* FINDAPLOTTEDANTITYRELATEDTO *);

    FUNCTION pos_of_entity_related_TO(ENTPT: ENTITYPTR ): COORDPTR;
    (* TO FIND THE POSITION OF AN ENTITY RELATED TO SPECIFIED ENTITY *)
    VAR EPPT: ENTPLOTPTR;
    BEGIN
        (* FIND A PLOTTED ENTITY or an average if this is desired*)
        If average_rel Then
            pos_of_entity_related_to := find_average_rel_pos ( entpt )
        Else Begin
            EPPT := FINDAPLOTTEDENTITYRELATEDTO( ENTPT );
            IF EPPT = NIL THEN pos_of_entity_related_TO := NULLCOORD
                          ELSE pos_of_entity_related_TO :=
                                   EPPT^.BOXPLOT^.CENTRE
        End (* Else *)
    END (* pos_of_entity_related_TO *);

    FUNCTION FINDBOXAROUNDCOORD( COORD: COORDPTR ): BOXPTR;
    (* TO FIND THE BOX WHICH ENCOMPASSES GIVEN COORD *)
    VAR EPPT: ENTPLOTPTR;
```

```
    BOXFND: BOOLEAN;
BEGIN
     EPPT := HDENTPLOT;
     BOXFND := FALSE;
     WHILE (EPPT <> NIL) AND (NOT BOXFND) DO
     WITH EPPT^.BOXPLOT  DO
          IF (XCOORDS[LESSXBND] <= COORD^.X) AND
             (XCOORDS[GTXBND] >= COORD^.X) AND
             (YCOORDS[LESSYBND] <= COORD^.Y) AND
             (YCOORDS[GTYBND] >= COORD^.Y)       THEN
               BOXFND := TRUE
           ELSE EPPT := EPPT^.NEXTENTRY;

      If eppt = Nil Then findboxaroundcoord := Nil
                    Else FINDBOXAROUNDCOORD := EPPT^.BOXPLOT
 END (* FINDBOXAROUNDCOORD *);


Function  grad_of_line( coord1, coord2: coordptr ): Real;
{ To find the gradient of line connecting two coords }
Var x1, y1, x2, y2: component;
    grad: Real;
Begin
     x1 := coord1 .x;
     x2 := coord2 .x;
     y1 := coord1 .y;
     y2 := coord2 .y;

     { Use formula grad = (y1-y2)/(x1-x2)

       First check for vertical lines, x1=x2, or lines close to
       vertical }
     If Abs(x1 - x2) < 1 Then grad := infinite_grad
                         Else grad := (y1 - y2)/(x1 - x2);

     grad_of_line := grad

End {grad_of_line};

Function find_const( grad: Real;
                     coord1: coordptr ): Real;
{ To find the constant of the line connecting two coorHds }
Begin
     { Use formula c = y - mx for line
       First check for infinite grad (ie. vertical line), in
       which case x = const. }

     If grad = infinite_grad Then find_const := coord1^.x
     Else
          find_const := coord1 .y - (grad * coord1^.x)

End {find_const};

Function find_rect_region( box1: boxptr;
                           other_coord: coordinate ): rectedge;
```

```
{ To find region other_coord is in relative to box1 }
Var c1, c2, c3, c4: Real;
    other_x, other_y: component;
    edge: rectedge;
Begin
     { Assume model has four quadrants corr. to area between line
       y = -x + (y1+x1) from vertex (x1 ,y1) &
       y = x + (y2-x2) from vertex (x2 , y2) etc.

       First set up consts. for each of four lines }

     With box1↑ do
     Begin
          c1 := ycoords[1] + xcoords[1];
          c2 := ycoords[2] - xcoords[2];
          c3 := ycoords[3] + xcoords[3];
          c4 := ycoords[4] - xcoords[4]
     End {With};

     other_x := other_coord.x;
     other_y := other_coord.y;

     { Try top first - assume if equal a line is in top or bottom regions
       rather than sides }

     If (other_y <= (c1-other_x)) And (other_y <= (other_x+c2)) Then
          edge := top
     Else If (other_y > (other_x+c2)) And (other_y > (c3-other_x)) Then
          edge := right
     Else If (other_y <= (c3-other_x)) And (other_y <= (other_x+c4)) Then
          edge := bottom
     Else { other_y > (other_x + c4) & other_y > (c1 - other_x) }
          edge := left;

     find_rect_region := edge

End { find_rect_region };

Function pnrect( px, py: Integer;
                 polyxcoords, polyycoords: rectcoord ): Integer;
{ This is an amendment of pnpoly taken from Computer Journal
  Vol 24 No. 1, p94. It determines if a given point (px,py) is
  inside (result = +1), on the edge of (result = 0) or outside
  (result = -1) the polygon og given coords (in this case a rectangle).
  To change to polygon use conformant array schemas }

Var result: Integer;
    lx, mx, nx, ly, my, ny: Boolean;
    xj, yj, xi, yi, i, j: Integer;
    test: Integer;

Begin
     { A vertical line is drawn thru point in question, if it crosses
       the polygon an odd no. of times, then the point is inside the
       polygon }
```

```
        result := -1;
        xj := polyxcoords[ 1 ] - px;
        yj := polyycoords[ 1 ] - py;

        For i := 1 to 4 Do
        Begin
             j := 1 + (i Mod 4);
             xi := xj;
             yi := yj;
             xj := polyxcoords[ j ] - px;
             yj := polyycoords[ j ] - py;
             lx := (xi < 0) And (xj < 0);
             mx := (xi > 0);
             nx := (xj > 0);
             ly := (yi < 0) And (yj < 0);
             my := (yi > 0);
             ny := (yj > 0);

             test := (((yi*xj) - (xi*yj)) * (xj-xi));

             If test = 0 Then
             Begin
                  If Not( lx Or ly Or (mx And nx) Or (my And ny)) Then
                       result := 0
             End
             Else If (test > 0) And (mx or nx) And
                     ((Not mx) Or (Not nx)) And (my or ny) Then
                  result := -result;

        End { For };

        pnrect := result

End { pnrect };

Function box_with_box_intersection( box1 , box2: boxtyp ): Boolean;
{ To find any intersection between entities, one plotted & one not }
Var i: Integer;
    intfnd: Boolean;
    box1_xcoords, box2_xcoords, box1_ycoords, box2_ycoords: rectcoord;
Begin
     { Try each point of unplotted entity to see if any of then occur
       within plotted entity }

     box1_xcoords := box1.xcoords;
     box2_xcoords := box2.xcoords;
     box1_ycoords := box1.ycoords;
     box2_ycoords := box2.ycoords;

     i := 1;
     intfnd := False;
     While (i <= 4) And (Not intfnd) Do
          If (pnrect( box1_xcoords[ i ] ,
                      box1_ycoords[ i ],
```

```
                    box2_xcoords,
                    box2_ycoords ) <= 0 ) Then
                intfnd := True
            Else
                i := i + 1;

    box_with_box_intersection := intfnd

End { box_with_box_intersection };

Function coord_not_in_range( coord1, coord2, intcoord: coordinate):
                                                    Boolean;
{ See if intcoord is outside of range from coord1 to coord2 }
Begin
    coord_not_in_range :=
        ((( coord1.x < intcoord.x ) And
          ( coord2.x < intcoord.x )    ) Or
         (( coord1.x > intcoord.x ) And
          ( coord2.x > intcoord.x )    )   ) Or
        ((( coord1.y < intcoord.y ) And
          ( coord2.y < intcoord.y )    ) Or
         (( coord1.y > intcoord.y ) And
          ( coord2.y > intcoord.y )    )   )
End { coord_not_in_range };

Function infinite_case( c1 , m2 , c2: Real ): coordptr;
{ To deal with case of finding intersection when line1 is
  infinite, ie. x = const }
Var coord: coordptr;
Begin
    New( coord );
    coord^.x := Round( c1 );
    If (m2 * c1) < Maxint Then coord^.y := Maxint
                          Else coord^.y := Round((m2*c1)+c2);

    infinite_case := coord
End { infinite_case };

Function line_intersection( m1, c1, m2, c2: Real): coordptr;
{ To find point of intersection of lines
    y = m1x + c1  and y = m2x + c2

  Use simultaneous eqns. to find
    x = (c1 - c2) / (m2 - m1)
  & use this value in y = m1x + c1.

  Must check for parallel lines first (m2 = m1) & for infinite grad
  (in which case (m2-m1) = infinity)      }

Var coord: coordptr;
    x_val: Real;
Begin
    If Abs( m1-m2 ) < 0.1 Then
        { parallel lines or close to, intersection at infinity }
        coord := infinity
```

```
        Else If m1 = infinite_grad Then
             coord := infinite_case( c1 , m2 , c2 )
        Else If m2 = infinite_grad Then
             coord := infinite_case( c2 , m1 , c1 )
        Else Begin
             New( coord );
             { Due to rounding errors, must keep full x value, else
               get wrong value when multiply by gradient }
             x_val := (c1-c2) / (m2-m1);
             coord^.x := Round( x_val ); {only want integer part}
             coord^.y := Round( (m1*x_val) + c1)
        End { Else };

        line_intersection := coord
   End {line_intersection};

   Function line_and_line_intersection( colst1, colst2: coordlistptr ):
                                        Boolean;
   { To see if two relations intersect }
   Var tcolst1, tcolst2: coordlistptr;
       intcrd: coordptr;
       intfnd: Boolean;
   Begin
        tcolst1 := colst1;
        intfnd := False;

        While (tcolst1 <> Nil) And (Not intfnd) Do
        Begin
             tcolst2 := colst2;
             While (tcolst2 <> Nil) And (Not intfnd) Do
             Begin
                  intcrd := line_intersection( tcolst1^.linegrad,
                                               tcolst1^.lineconst,
                                               tcolst2^.linegrad,
                                               tcolst2^.lineconst );
                  If coord_not_in_range( tcolst1^.startcoord^,
                                         tcolst1^.endcoord^,
                                         intcrd^              ) Or
                     coord_not_in_range( tcolst2^.startcoord^,
                                         tcolst2^.endcoord^,
                                         intcrd^              ) Then
                  Begin
                       If intcrd <> infinity Then Dispose( intcrd );
                       intcrd := nointersection
                  End;

                  If intcrd = nointersection Then intfnd := True
                  Else
                       tcolst2 := tcolst2^.nextcoord
             End {While};

             If Not intfnd Then tcolst1 := tcolst1^.nextcoord
        End {While};
        line_and_line_intersection := intfnd
   End {line_and_line_intersection};
```

```
Function no_of_intersecting_rels( colstl: coordlistptr ): Integer;
{ To calculate the no. of rels. which intersect with given line }
Var rppt: relplotptr;
    no_rels: Integer;
Begin
     { This is only used to calulate penalties }
     rppt := hdrelplot;
     no_rels := 0;

     While (rppt <> Nil) Do
     Begin
          If line_and_line_intersection( colstl , rppt^.crdlist ) Then
          Begin
              { What a greater penalty for crossing digonal line }
              If (rppt^.crdlist^.linegrad = 0) Or
                 (rppt^.crdlist^.linegrad = infinite_grad) Then
                 { is equivalent of crossing 10 straight lines }
                 no_rels := no_rels + 10
              Else
                 no_rels := no_rels + 1
          End;
          rppt := rppt^.nextentry
     End {While};

     no_of_intersecting_rels := no_rels
End {no_of_intersecting_rels};

Function rel_in_same_path( colstl: coordlistptr ): Boolean;
{ To see if there is a rel. in the same path as given one }
Var rppt: relplotptr;
    clp: coordlistptr;
    relfnd: Boolean;
Begin
     relfnd := False;
     rppt := hdrelplot;
     While (rppt <> Nil) And (Not relfnd) Do
     Begin
          clp := rppt^.crdlist;
          While (clp <> Nil) And (Not relfnd) Do
          Begin
               If (clp^.linegrad = colstl^.linegrad) And
                  (clp^.lineconst = colstl^.lineconst) Then
                    relfnd :=
                       (Not coord_not_in_range( clp^.startcoord^,
                                                clp^.endcoord^,
                                                colstl^.startcoord^))
                    Or (Not coord_not_in_range( clp^.startcoord^,
                                                clp^.endcoord^,
                                                colstl^.endcoord^))
                    Or (Not coord_not_in_range( colstl^.startcoord^,
                                                colstl^.endcoord^,
                                                clp^.startcoord^))
                    Or (Not coord_not_in_range( colstl^.startcoord^,
                                                colstl^.endcoord^,
```

```
                                                    clp↑.endcoord↑));
                If Not relfnd Then clp := clp↑.nextcoord
            End {While};

            If Not relfnd Then rppt := rppt↑.nextentry
        End {While};

        rel_in_same_path := relfnd
    End { rel_in_same_path };

    Function box_intersection( ml, cl: Real;
                               strt_coord: coordptr;
                               box: boxptr ): coordptr;
    { To find the intersection between a box & the line with
      eqn. y = mlx + cl }
    Var less_x_bound, gt_x_bound, less_y_bound, gt_y_bound: component;
        intersect_coord: coordptr;
        i: Integer;
        edge: rectedge;
    Begin
        { Take each edge in turn & see if any intersection within
          bounds of box vertices }

        With box↑ Do
        Begin
            less_x_bound := xcoords[ lessxbnd ];
            gt_x_bound := xcoords[ gtxbnd ];
            less_y_bound := ycoords[ lessybnd ];
            gt_y_bound := ycoords[ gtybnd ]
        End { With };

        { See if intersection with any edges }
        i := 0;
        edge := find_rect_region( box , strt_coord↑ );

        Repeat
            Case edge Of
                top: intersect_coord :=
                    line_intersection( ml, cl, 0, gt_y_bound );
                right: intersect_coord :=
                    line_intersection( ml, cl, infinite_grad, gt_x_bound);
                bottom: intersect_coord :=
                    line_intersection( ml, cl, 0, less_y_bound );
                left: intersect_coord :=
                    line_intersection( ml,cl,infinite_grad,less_x_bound );
            End {Case};

            If (intersect_coord↑.y < less_y_bound) Or
               (intersect_coord↑.y > gt_y_bound) Or
               (intersect_coord↑.x < less_x_bound) Or
               (intersect_coord↑.x > gt_x_bound)     Then
            Begin
                If intersect_coord <> infinity Then
                    Dispose( intersect_coord );
                intersect_coord := nointersection;
```

```
                If edge = left Then edge := top
                               Else edge := Succ( edge );
                i := i + 1
            End { Then };

        Until ( i = 4 ) Or ( intersect_coord <> nointersection );

        { resultant intersection is the intersect coord found }

        box_intersection := intersect_coord
    End {box_intersection};

    Function line_intersects_entity( m1,c1: Real;
                                     coord1, coord2: coordptr;
                                     box1, box2: boxptr ):
                                                        Boolean;
    { To see if given line intersects with any entities }
    Var eppt: entplotptr;
        intersect_coord: coordptr;
    Begin
        { Check every plotted entity to see if there is any intersection¾
        eppt := hdentplot;
        intersect_coord := nointersection;

        While (eppt <> Nil) And (intersect_coord = nointersection) Do
        With eppt↑ Do
        If (boxplot = box1) Or (boxplot = box2) Then eppt := nextentry
        Else Begin
            intersect_coord := box_intersection( m1, c1, coord1, boxplot );

            If (intersect_coord <> nointersection) Then
               If
               coord_not_in_range( coord1¼ , coord2¼ , intersect_coord¼ )
            Then Begin
                { intersection is outside of boundary of line ¾
                If intersect_coord <> infinity Then
                    Dispose( intersect_coord );
                intersect_coord := nointersection
            End;

            If (intersect_coord = nointersection) Then eppt := nextentry
        End {While With Else};

        { If pointer is now nil, all plotted entities have been tested &
          none cause intersection }
        line_intersects_entity :=
                                (intersect_coord    nointersection)
    End {line_intersects_entity¼;

    Function entity_intersects_entity( box: boxptr ): Boolean;
    { To see if there are any entities which entity would overlap with
      at given coord }
    Var eppt: entplotptr;
        intfnd: Boolean;
    Begin
```

```
     { Examine every plotted entity to see if there are any which
       overlap }
     eppt := hdentplot;
     intfnd := False;

     While (eppt <> Nil) And (Not intfnd) Do
          If box_with_box_intersection( box^ , eppt^.boxplot^ ) Then
               intfnd := True
          Else
               eppt := eppt^.nextentry;

     entity_intersects_entity := intfnd
End {entity_intersects_entity};

Function no_relation_intersection( colst: coordlistptr;
                                   box: boxptr ): Boolean;
{ To see if the box intersects with given relation plot }
Var tcolst: coordlistptr;
    intersect_coord: coordptr;
Begin
     { Try each relation plot line to see if any intersection }
     tcolst := colst;
     intersect_coord := nointersection;

     While (tcolst <> Nil) And (intersect_coord = nointersection) Do
     With tcolst^ Do
     Begin

          intersect_coord := box_intersection( linegrad,
                                               lineconst,
                                               startcoord,
                                               box );
          If (intersect_coord <> nointersection) Then If
            coord_not_in_range( endcoord^, startcoord^, intersect_coord^)
          Then Begin
               { Intersection is outside of boundary of line }
               If intersect_coord <> infinity Then
                    Dispose( intersect_coord );
               intersect_coord := nointersection
          End;

          If intersect_coord = nointersection Then
               tcolst := nextcoord
     End {While With};

     { If come to end of list, can't be any intersection }
     no_relation_intersection := (intersect_coord = nointersection)
End {no_relation_intersection};

Function is_rel_ent_intersects_with( box:boxptr ): Boolean;
{ To see if there are any relations that an entity intersects with if
  placed at given position }
Var rppt: relplotptr;
    intfnd: Boolean;
Begin
```

```
dices

      { Take each relation in turn to see if there is any which
        intersect with box }
      rppt := hdrelplot;
      intfnd := false;

      While (rppt <> Nil) And (Not intfnd) Do
            If Not no_relation_intersection( rppt^.crdlist , box ) Then
                  intfnd := True

            Else
                  rppt := rppt^.nextentry;

      { If come to end of list of plotted relations, then no intersect }
      is_rel_ent_intersects_with := intfnd
End {is_rel_ent_intersects_with};

Function free_position( box: boxptr ): Boolean;
{ To see if coord is free for a box }
Var result: Boolean;
Begin
      result := Not is_rel_ent_intersects_with( box );
      If result Then
            result := Not entity_intersects_entity( box );

      free_position := result
End {free_position};

Function closest_entity( dir: linedirection;
                         comp,
                         other_comp: component;
                         cnstcomp: component;
                         strt_coord: coordptr;
                         box1,box2: boxptr ):  component;
{ To find closest entity to comp in the direction of other_comp }
Var dir_of_box: rectedge;
    best_line_lth: Integer;
    cmp, best_comp: component;
    grad, cnst: Real;
    eppt: entplotptr;
    intersect_coord: coordptr;
Begin
      { First find actual line direction & eqn. }
      If dir = vert Then
      Begin
            { Is a vertical line thru comp }
            grad := infinite_grad;
            If comp < other_comp Then dir_of_box := top
                                 Else dir_of_box := bottom

      End
      Else Begin
            { Is a horizontal line thru comp }
            grad := 0;
            If comp < other_comp Then dir_of_box := right
                                 Else dir_of_box := left

      End {Else};
```

```
        cnst := cnstcomp;

        { Search complete plotted entity list for closest entity }

        eppt := hdentplot;
        best_line_lth := Maxint;
        best_comp := Maxint;

        While eppt <> Nil Do
        With eppt^.boxplot^ Do
        Begin
             { See if possible that line will intersect }
             If ( (box1 <> eppt^.boxplot) And
                  (box2 <> eppt^.boxplot)     ) And
                (((dir = vert) And (xcoords[gtxbnd] >= cnst)
                  And (xcoords[lessxbnd] <= cnst) And
                  (((dir_of_box = top) And (centre^.y < comp)) Or
                   ((dir_of_box = bottom) And (centre^.y <= comp))) ) Or
                 ((dir = hor) And (ycoords[gtybnd] >= cnst)
                  And (ycoords[lessybnd] < cnst) And
                  (((dir_of_box = right) And (centre^.x >= comp)) Or
                   ((dir_of_box = left) And (centre^.x <= comp))    ) ))
              Then Begin
                   { Will intersect with box, question is is it closer }
                   intersect_coord := box_intersection( grad,
                                                        cnst,
                                                        strt_coord,
                                                        eppt^.boxplot );
                   If intersect_coord <> nointersection Then
                   Begin
                        If dir = vert Then cmp := intersect_coord^.y
                                      Else cmp := intersect_coord^.x;
                        If Abs(cmp - comp) < best_line_lth Then
                        Begin
                             best_line_lth := Abs(cmp - comp);
                             best_comp := cmp
                        End {If};
                        Dispose( intersect_coord )
                   End {If}
               End {If};

             eppt := eppt^.nextentry
        End {While};

        closest_entity := best_comp
   End {closest_entity};

   Function same_coord( coord1, coord2: coordinate ): Boolean;
   { To see if the two coords have the same value }
   Begin
        same_coord := (coord1.x = coord2.x) And
                      (coord1.y = coord2.y)
   End {same_coord};
```

```
PROCEDURE INITIALISE;

PROCEDURE SORTENTITIES;
(* PLACE THE ENTITIES INTO SOME ORDER. ENTITIES AT END OF LIST WILL HAVE
   ALL THEIR RELATIONSHIPS CONNECTED FIRST, BUT WILL BE IN THE-MOST OUT
   OF THE WAY POSITION *)
VAR ENTPT: ENTITYPTR;

    FUNCTION FINDNOOFRELS( ENTPT: ENTITYPTR): INTEGER;
    (* FIND THE NO OF RELATIONSHIPS OF THE ENTITY *)
    VAR ERPT: ENTITYRELPTR;
        NOOFRELS: INTEGER;
    BEGIN
         NOOFRELS := 0;
         ERPT := ENTPT↑.RELATONSPTR;
         WHILE ERPT <> NIL DO
         BEGIN
              NOOFRELS := NOOFRELS + 1;
              ERPT := ERPT↑.NEXTRELFORENTITY
         END (* WHILE *);
         FINDNOOFRELS := NOOFRELS
    END (* FINDNOOFRELS *);

    PROCEDURE PLACEINCHAIN( ENTPT: ENTITYPTR;
                            NORELS: INTEGER );
    (* PLACE THE SPECIFIED ENTITY IN THE SORTED ENTITY CHAIN IN ORDER OF
       INCREASING NO. OF RELATIONS *)
    VAR SRTDENTPT: ENTPLOTPTR;
        TMPSRTDENTPT: ENTPLOTPTR;
        ENTFND: BOOLEAN;
    BEGIN
         (* CREATE ENTRY *)
         IF HDSORTEDENT = NIL THEN
         BEGIN
              NEW( HDSORTEDENT );
              SRTDENTPT := HDSORTEDENT;
              SRTDENTPT↑.NEXTENTRY := NIL
         END
         ELSE BEGIN
              ENTFND := FALSE;
              TMPSRTDENTPT := HDSORTEDENT;
              WHILE (TMPSRTDENTPT↑.NEXTENTRY <> NIL) AND (NOT ENTFND) DO
                   IF sorted Then
                   Begin
                        If (ascending And
                        (TMPSRTDENTPT↑.NEXTENTRY↑.NOOFRELS > NORELS))
                        Or (Not ascending And
                                (tmpsrtdentpt↑.nextentry↑.noofrels < Norels))
                                Then entfnd := True
                        Else tmpsrtdentpt := tmpsrtdentpt↑.nextentry
                   End Else
                        TMPSRTDENTPT := TMPSRTDENTPT↑.NEXTENTRY;
              NEW( SRTDENTPT );
              IF sorted And (TMPSRTDENTPT = HDSORTEDENT) AND
                 ((ascending And (hdsortedent↑.noofrels > norels)) Or
```

```
                (Not ascending And (HDSORTEDENT↑.NOOFRELS < NORELS))) THEN
            BEGIN
                (* ADD NEW HEAD ENTRY *)
                SRTDENTPT↑.NEXTENTRY := HDSORTEDENT;
                HDSORTEDENT := SRTDENTPT
            END
            ELSE BEGIN
                (* ADD ENTRY IN MIDDLE OF LIST *)
                SRTDENTPT↑.NEXTENTRY := TMPSRTDENTPT↑.NEXTENTRY;
                TMPSRTDENTPT↑.NEXTENTRY := SRTDENTPT
            END (* ELSE *)
        END (* ELSE *);
        (* ADD VALUES OF ENTRY *)
        WITH SRTDENTPT↑ DO
        BEGIN
            ENTITY := ENTPT;
            NOOFRELS := NORELS;
            BOXPLOT := NIL
        END (* WITH SRTDENTPT *)
    END (* PLACEINCHAIN *);

BEGIN
    (* GO DOWN LIST OF ENTITES & PLACE THEM IN SORTED LIST IN ORDER OF
       NO. OF RELATIONSHIPS *)
    ENTPT := ENTITYLIST;
    WHILE ENTPT <> NIL DO
    BEGIN
        (* PUT IN LIST *)
        PLACEINCHAIN( ENTPT , FINDNOOFRELS( ENTPT ) );
        (* GET NEXT ENTITY *)
        ENTPT := ENTPT↑.NEXTENTITYPTR
    END (* WHILE *)
END (* SORTENTITIES *);

PROCEDURE CREATEUNPLOTTEDRELLIST;
(* TO CREATE A LIST OF UNPLOTTED RELATIONSHIPS *)
VAR RELPT: RELATONPTR;
BEGIN
    (* GO THROUGH LIST OF RELATIONSHIPS AND CREATE AN ENTRY POINTING AT
       EACH ONE, INITIALLY NO RELATIONS ARE PLOTTED *)
    RELPT := RELATONLIST;
    WHILE RELPT <> NIL DO
    BEGIN
        IF HDRELUNPLOT = NIL THEN
        (* CREATE FIRST ENTRY *)
        BEGIN
            NEW( HDRELUNPLOT );
            TLRELUNPLOT := HDRELUNPLOT
        END
        ELSE BEGIN
            (* CREATE A NEW ENTRY AT TAIL OF LIST *)
            NEW( TLRELUNPLOT↑.NEXTENTRY );
            TLRELUNPLOT := TLRELUNPLOT↑.NEXTENTRY
        END (* ELSE *);
        (* SET UP POINTER TO POINT AT APP. RELATION ENTRY *)
```

```
          WITH TLRELUNPLOT↑ DO
          BEGIN
               RELATON := RELPT;
               CRDLIST := NIL;
               NEXTENTRY := NIL
          END (* WITH *);
          (* GET NEXT RELATION ENTRY *)
          RELPT := RELPT↑.NEXTRELPTR
     END (* WHILE *)
END (* CREATEUNPLOTTEDRELLIST *);

BEGIN
      (* INITIALISATION CODE *)
      ENTITYLIST := NIL;
      TAILENTITYLIST := NIL;
      RELATONLIST := NIL;
      TAILRELATONLIST := NIL;
      INFILNAME := '        ';

     NEW( orign );
     orign↑.X := 0;
     orign↑.Y := 0;

     New( strtpt );
     strtpt↑.x := 0;
     strtpt↑.y := 0;

     NEW( INFINITY );
     INFINITY↑.X := MAXINT;
     INFINITY↑.Y := MAXINT;

     HDENTPLOT := NIL;
     HDSORTEDENT := NIL;
     HDRELPLOT := NIL;
     HDRELUNPLOT := NIL;
     LASTBOXBELOW := TRUE;

     readstuff;
     SORTENTITIES;
     CREATEUNPLOTTEDRELLIST
END (* SEGMENT INITIALISE *);

PROCEDURE PLACEENTITIES;
(* TO ARRANGE THE PLOTTING OF ALL ENTITIES AND THEIR CONNECTING RELS *)

VAR MAINEPPT: ENTPLOTPTR;
    TBOX: BOXPTR;
FUNCTION PLOTRELATIONSHIP( BOX1, BOX2: BOXPTR ): COORDLISTPTR;
(* TO PLOT A PATH FROM FIRST ENTRY TO SECOND ENTRY *)
VAR DIRCRDSTART, crdstart2: COORDPTR;
    VERTCOORDLST, HORCOORDLST, CRDLST: COORDLISTPTR;
    best_so_far,NOHOR, NOVERT, NODIR: INTEGER;
    STEDGE: RECTEDGE;
    OVERALLDIR: LINEDIRECTION;
    NOOPLINEDRAWN, SECONDTRY: BOOLEAN;
```

```
STRTBOX: BOXPTR;

Procedure dispose_list_entry( lstpt: coordlistptr );
{To dispose of given list entry}
Begin
     If lstpt↑.endcoord <> Nil Then Dispose( lstpt↑.endcoord );
     Dispose( lstpt )
End {dispose_list_entry};

Procedure dispose_list( stpt: coordlistptr );
{ To dispose of a coordinate list }
Var lstpt: coordlistptr;
Begin
     If stpt <> Nil Then
          {Dispose of first startcoord}
          If stpt↑.startcoord <> Nil Then Dispose( stpt↑.startcoord );
     While stpt <> Nil Do
     Begin
          lstpt := stpt;
          stpt := stpt↑.nextcoord;
          dispose_list_entry( lstpt )
     End {While}
End {dispose_list};

PROCEDURE MARKCOORDTAKEN( CLP: COORDLISTPTR;
                          COORD: COORDPTR;
                          EDGE: RECTEDGE;
                          BOX: BOXPTR     );
(* TO MARK GIVEN COORD ON BOX EDGES AS TAKEN *)
VAR COX, COY: COMPONENT;
    I: INTEGER;
BEGIN
     COX := COORD↑.X;
     COY := COORD↑.Y;
     (* FIND EDGE COORD IS ON IN RELATION TO CENTRE & SET UPPER ARRAY
        ENTRY ACCORDING TO COORD *)

     WITH BOX↑ DO
     BEGIN
          CASE EDGE OF
               TOP , BOTTOM:
                    RELSONEDGE[ EDGE ,
                       ((COX-XCOORDS[LESSXBND]) DIV GAPBETRELS) ]:= CLP;
               LEFT , RIGHT:
                    RELSONEDGE[ EDGE ,
                       ((COY-YCOORDS[LESSYBND]) DIV GAPBETRELS) ]:= CLP
          END (* CASE *)
     END (* WITH *)
END (* MARKCOORDTAKEN *);

Function posn_is_free( comp: component;
                       EDGE: RECTEDGE;
                       BOX: BOXPTR     ): Boolean;
{ To see if given position on edge of box is free }
Var result: Boolean;
```

```
BEGIN

     WITH BOX[ DO
     BEGIN
          CASE EDGE OF
               TOP , BOTTOM:
                    result := RELSONEDGE[ EDGE ,
                    ((comp-XCOORDS[LESSXBND]) DIV GAPBETRELS) ] = free;
               LEFT , RIGHT:
                    result := RELSONEDGE[ EDGE ,
                    ((comp-YCOORDS[LESSYBND]) DIV GAPBETRELS) ] = free
          END (* CASE *)
     END (* WITH *);
     posn_is_free := result
END (* MARKCOORDTAKEN *);

FUNCTION FINDPOSN( startpos, endpos, inc: INTEGER;
                   EDGERELS: EDGESTARTS ): INTEGER;
(* FIND FIRST FREE POSITION STARTING FROM CENTRE *)
VAR POS: INTEGER;
    POSFND: BOOLEAN;
BEGIN
     POS := startpos;
     POSFND := FALSE;
     WHILE (POS <> endpos) AND (NOT POSFND) DO
          IF (EDGERELS[ POS ] = FREE) THEN POSFND := TRUE
          ELSE
               POS := POS + inc;

     If Not posfnd Then pos := Maxint;
     FINDPOSN := POS
END (* FINDPOSN *);

FUNCTION FINDCENTREPOSN( MAXNORELS: INTEGER;
                         EDGERELS: EDGESTARTS ): INTEGER;
(* FIND FIRST FREE POSITION CLOSEST TO CENTRE *)
VAR LPOS, RPOS: INTEGER;
    CENTRE: INTEGER;
    centreposn: INTEGER;
BEGIN
     CENTRE := MAXNORELS DIV 2;
     LPOS := FINDPOSN( centre, 0, -1, EDGERELS );

     IF LPOS = CENTRE THEN CENTREPOSN := CENTRE
     ELSE BEGIN
          (* ATTEMPT SEARCH DONW RIGHT SIDE *)
          RPOS := FINDPOSN( centre, maxnorels, 1, EDGERELS );

          (* FIND WHICH OF LPOS & RPOS IS CLOSEST TO CENTRE *)

          IF ABS(CENTRE-RPOS) <= ABS(CENTRE-LPOS) THEN
               CENTREPOSN := RPOS
          ELSE
               CENTREPOSN := LPOS
     END (* ELSE *);
```

```
    FINDCENTREPOSN := CENTREPOSN
END (* FINDCENTREPOSN*);

FUNCTION GETRELEND( RELSONEDGE: EDGERELSTARTS;
                    EDGE: RECTEDGE;
                    OTHERCOORD: COORDINATE;
                    CENTRE: COORDINATE): COORDPTR;
(* TO FIND A FREE PART OF GIVEN EDGE CLOSEST TO GIVEN COORD.
   IF NO FREE SPACE, GET A VERTEX.
   SO HAVE MANY RELS, ALL AT VERTEX *)
VAR ENDREL: COORDPTR;
    ENDRELPOSN: INTEGER;
    MAXNORELS: INTEGER;
    CENTREVAL: COMPONENT;
    OTHERCOMP: COMPONENT;
BEGIN
    (* SET UP ENDREL COORD *)
    NEW( ENDREL );

    (* DEAL WITH SEPARATE PROCESSING NEED FOR X EDGES & Y EDGES *)
    CASE EDGE OF
        TOP, BOTTOM: BEGIN
            MAXNORELS := MAXNOXRELS;
            CENTREVAL := CENTRE.X;
            OTHERCOMP := OTHERCOORD.X;
            ENDREL↑.X := MAXINT; (* TO DENOTE NOT YET FOUND *)
            IF EDGE = TOP THEN ENDREL↑.Y := CENTRE.Y + HALFYGAP
                          ELSE ENDREL↑.Y := CENTRE.Y - HALFYGAP
        END (* BEGIN *);
        LEFT, RIGHT: BEGIN
            MAXNORELS := MAXNOYRELS;
            CENTREVAL := CENTRE.Y;
            OTHERCOMP := OTHERCOORD.Y;
            ENDREL↑.Y := MAXINT;
            IF EDGE = RIGHT THEN ENDREL↑.X := CENTRE.X + HALFXGAP
                            ELSE ENDREL↑.X := CENTRE.X - HALFXGAP
        END (* BEGIN *)
    END (* CASE *);

    (* FIND RELATIVE POSITION ALONG LINE OF START *)
    IF OTHERCOMP = CENTREVAL THEN
        (* TRY & FIND POSITION STARTING FROM MIDDLE *)
        ENDRELPOSN := FINDCENTREPOSN( MAXNORELS , RELSONEDGE[ EDGE ] )
    ELSE IF OTHERCOMP < CENTREVAL THEN
        (* START FROM CENTRE DOWN LHS TO FIND POSN *)
        ENDRELPOSN := FINDPOSN( 1,(maxnorels div 2)+1, 1,
                                RELSONEDGE[ EDGE ] )
    ELSE (* START FROM CENTRE & WORK DOWN RHS *)
        ENDRELPOSN := FINDPOSN( maxnorels-1,(maxnorels div 2)-1, -1,
                                RELSONEDGE[ EDGE ] );

    (* SET APPROPRIATE VALUE OF COORD *)
    IF (ENDRELPOSN >= 1) AND (ENDRELPOSN <= MAXNORELS) THEN
        (* SPACE ON EDGE, SO CAJ SET COORD TO POSN *)
        CASE EDGE OF
```

```
            TOP, BOTTOM:
               ENDREL↑.X:=(CENTREVAL-HALFXGAP)+(ENDRELPOSN*GAPBETRELS);
            LEFT, RIGHT:
               ENDREL↑.Y :=(CENTREVAL-HALFYGAP)+(ENDRELPOSN*GAPBETRELS)
         END (* CASE *);
   GETRELEND := ENDREL
END (* GETRELEND *);

Function find_direct_link_end( coordl: coordptr;
                               stedge: rectedge;
                               edge: rectedge;
                               boxa,boxb: boxptr ): coordptr;
{ To find a position for a direct link on an edge }
Var othercoord: coordptr;
Begin
   othercoord := Nil;
   { First see if a horizontal or vertical line is possible }
   If (((stedge = top) And (edge = bottom)) Or
       ((stedge = bottom) And (edge = top))) And
      (coordl↑.x < boxb↑.xcoords[gtxbnd]) And
      (coordl↑.x > boxb↑.xcoords[lessxbnd]) Then
   Begin
      If posn_is_free( coordl↑.x , edge , boxb ) Then
      Begin
         New (othercoord);
         othercoord↑.x := coordl↑.x;
         If edge = top Then
            othercoord↑.y := boxb↑.centre¼.y + halfygap
         Else othercoord↑.y := boxb↑.centre¼.y - halfygap
      End
    End
    Else If (((stedge = left) And (edge = right)) Or
             ((stedge = right) And (edge = left)) ) And
            (coordl↑.y < boxb↑.ycoords[gtybnd]) And
            (coordl↑.y > boxb↑.ycoords[lessybnd]) Then
    Begin
       If posn_is_free( coordl↑.y , edge , boxb ) Then
       Begin
          New ( othercoord );
          othercoord↑.y := coordl¼.y;
          If edge = right Then
             othercoord↑.x := boxb↑.centre↑.x + halfxgap
          Else othercoord↑.x := boxb↑.centre↑.x - halfxgap
       End
    End;

    If othercoord = Nil Then
       othercoord := getrelend( boxb↑.relsonedge ,
                                edge ,
                                coordl↑ ,
                                boxb↑.centre↑ );
    find_direct_link_end := othercoord
End {find_direct_link_end};
```

```
FUNCTION SPACEFOUND( COORD: COORDPTR ): BOOLEAN;
(* TO SEE IF SPACE ON EDGE AT COORD *)
VAR RESULT: BOOLEAN;
BEGIN
     RESULT := TRUE;
     IF COORD = NIL THEN RESULT := FALSE
     ELSE IF (COORD↑.X = MAXINT) OR (COORD↑.Y = MAXINT) THEN
          RESULT := FALSE;

     SPACEFOUND := RESULT
END (* SPACEFOUND *);

FUNCTION FINDBESTPOS( DIR: LINEDIRECTION;
                      COMP,OTHERCOMP,CNSTCOMP: COMPONENT;
                      STRTCOORD: COORDPTR;
                      BOX1, BOX2: BOXPTR ): COMPONENT;
(* FIND THE BEST LINE IN GIVEN DIRECTION WHICH COMES CLOSEST TO OTHER
   COMP *)
VAR BESTCOMP: COMPONENT;
    CLOSESTINTERSECTION: COMPONENT;
    LENGTH: INTEGER;
BEGIN
     (* FIRST FIND CLOSEST ENTITY WHICH INTERSECTS WITH LINE *)
     CLOSESTINTERSECTION := closest_entity( DIR , COMP, OTHERCOMP,
                                            CNSTCOMP,STRTCOORD,
                                            BOX1, BOX2 );

     (* SEE IF THIS IF FURTHER THEN OTHER COMPONENT.
        IF IT IS THEN CAN DRAW A LINE ALL THE WAY TO THE
        OTHER COMP.- THIS ALL DEPENDS ON WHETHER LINE IS GOING
        TO LEFT/BOTTOM OR RIGHT/TOP *)
     IF OTHERCOMP > COMP THEN
     BEGIN
          (* GOING RIGHT / TOP *)
          IF (CLOSESTINTERSECTION >= OTHERCOMP) THEN
               (* THE INTERSECTION WITH ENTITY IS FURTHER AWAY THAN
                  LONGEST LINE WOULD LIKE TO DRAW *)
               BESTCOMP := OTHERCOMP
          ELSE
               (* JUST DRAW A LINE TO SOME POINT AS LONG AS POSSIBLE
                  WITHOUT CROSSING ENTITY & LEAVING A REASONABLE GAP *)
               BESTCOMP := COMP +
                           ((CLOSESTINTERSECTION-COMP) DIV PROPORTION );
     END (* GOING RIGHT *)
     ELSE
          (* GOING LEFT / BOTTOM *)
          IF (CLOSESTINTERSECTION = MAXINT) (* NO INTERSECTION *) OR
             (CLOSESTINTERSECTION <= OTHERCOMP) THEN
               BESTCOMP := OTHERCOMP
          ELSE
               BESTCOMP := COMP -
                           ((COMP-CLOSESTINTERSECTION) DIV PROPORTION);
     FINDBESTPOS := BESTCOMP
```

```
     END (* FINDBESTPOS *);

     PROCEDURE SINGLELINE( PRESPATH: COORDLISTPTR );
(* TO DEAL WITH A SINGLE LINE BETWEEN ENTITIES *)
BEGIN
     WITH PRESPATH↑ DO
     BEGIN
          LINEGRAD := grad_of_line( STARTCOORD , ENDCOORD );
          LINECONST := find_const( LINEGRAD , STARTCOORD );
     END (* WITH *)
END (* SINGLELINE *);

PROCEDURE ENDNOTSTRAIGHT( DIR: LINEDIRECTION;
                          PRESPATH, PREVPATH: COORDLISTPTR;
                          OLDCO: COORDINATE;
                          PRESCO1, PRESCO2,
                          PREVCO1, PREVCO2: COORDPTR;
                          BOX1, BOX2: BOXPTR;
                          VAR NUM: INTEGER );
(* TO SORT OUT END OF PATH WHEN END NOT IN LINE WITH START *)
VAR LINEPOSS: BOOLEAN;
    TCO: COORDINATE;
    COMP, OTHERCOMP: COMPONENT;
BEGIN
     IF PREVPATH = NIL THEN SINGLELINE( PRESPATH )
     ELSE BEGIN
          WITH PRESPATH↑ DO
          BEGIN
               TCO := PRESCO1↑;
               IF DIR = HOR THEN
               BEGIN
                    LINECONST := PRESCO2↑.Y;
                    PRESCO1↑.Y := PRESCO2↑.Y;
                    COMP := PRESCO1↑.X;
                    OTHERCOMP := PRESCO2↑.X
               END
               ELSE BEGIN
                    LINECONST := PRESCO2↑.X;
                    PRESCO1↑.X := PRESCO2↑.X;
                    COMP := PRESCO1↑.Y;
                    OTHERCOMP := PRESCO2↑.Y
               END (* ELSE *);

               (* FIRST SEE IF LINE IS POSSIBLE *)
               LINEPOSS := (FINDBESTPOS(DIR , COMP, OTHERCOMP,
                                        ROUND(LINECONST),
                                        PRESCO1,
                                        (* GLOBAL BOXES *) BOX1, BOX2 )

                                        = OTHERCOMP );
          END (* WITH *);

          IF LINEPOSS THEN
          BEGIN
               (* SET PREV. LINE TO MATCH NEW START COORD *)
```

```
        (* FIRST SEE IF LINE SHORTER *)
        IF ((PREVPATH↑.LINEGRAD = 0) AND
           ( ABS(PREVCO1↑.X - PREVCO2↑.X) >=
             ABS(PREVCO1↑.X - PRESCO1↑.X))) OR
           ((PREVPATH↑.LINEGRAD = infinite_grad) AND
           ( ABS(PREVCO1↑.Y - PREVCO2↑.Y) >=
             ABS(PREVCO1↑.Y - PRESCO1↑.Y)))
        THEN
             (* IS SHORTER SO CAN JUST SET NEW COORD *)
             PREVCO2↑ := PRESCO1↑
        ELSE BEGIN
             (* LINE LONGER, SO MUST ENSURE CAN EXTEND *)
             IF PREVPATH↑.LINEGRAD = 0 THEN
             BEGIN
                  COMP := PREVCO1↑.X;
                  OTHERCOMP := PRESCO1↑.X
             END
             ELSE BEGIN
                  COMP := PREVCO1↑.Y;
                  OTHERCOMP := PRESCO1↑.Y
             END;
             LINEPOSS := (FINDBESTPOS(DIR , COMP, OTHERCOMP,
                                 ROUND(PREVPATH↑.LINECONST),
                                 PREVCO1,
                                 (* GLOBAL BOXES *) BOX1, BOX2 )

                                 = OTHERCOMP );
             IF LINEPOSS THEN
                  (* ALL OK, SO CAN EXTEND *)
                  PREVCO2↑ := PRESCO1↑
        END (* ELSE *)
   END (* THEN *);
   IF NOT LINEPOSS THEN
   WITH PRESPATH↑ DO
   BEGIN
        (* CAN'T CHANGE LINE, SO RESET, TEST IS VIABLE & SET CONST &
           GRAD *)
        (* INCREASE SCORE, AS NOT AS GOOD A LINE AS POSS. *)
        NUM := NUM + 1;
        PRESCO1↑ := TCO;
        LINEGRAD := grad_of_line( PRESCO1 , PRESCO2 );
        LINECONST := find_const( LINEGRAD , PRESCO1 );
        IF line_intersects_entity( LINEGRAD ,
                                   LINECONST ,
                                   PRESCO1 ,
                                   PRESCO2 ,
                                   BOX1 , BOX2  ) THEN
        BEGIN
             (* MORE SERIOUS PROBLEM, CAN'T DRAW THIS LINE *)
             NUM := NUM + 10;
             (* ORIGINALLY HEADED FOR OLDEND & THIS WAS VIABLE, SO
                SET TO THIS *)
             PRESCO2↑ := OLDCO;
             LINEGRAD := grad_of_line( PRESCO1 , PRESCO2 );
             LINECONST := find_const( LINEGRAD , PRESCO1 );
```

```
                END (* THEN *)
            END (* THEN WITH *)
        END (* PREVPATH   NIL *)
END (* ENDNOTSTRAIGHT *);

PROCEDURE ENDPATH( DIR: LINEDIRECTION;
                   PRESPATH, PREVPATH: COORDLISTPTR;
                   BOX: BOXPTR;
                   VAR NUM: INTEGER );
(* TO END A PATH *)
BEGIN
     WITH PRESPATH↑ DO
     BEGIN
          STRORFIN := TRUE;
          NEXTCOORD := NIL;
          ENDEDGE := find_rect_region( BOX , STARTCOORD↑ );
          ENDCOORD := GETRELEND( BOX↑.RELSONEDGE , ENDEDGE ,
                                 STARTCOORD↑ , BOX↑.CENTRE↑ );
          IF NOT SPACEFOUND( ENDCOORD ) THEN
          BEGIN
               (* NO SPACE FOR DESIRED COORD, SO DOUBLE UP *)
               (* BAD SITUATION, SO INCREASE SCORE *)
               NUM := NUM + 10;
               (* MUST BE EITHER HOR. OT VERT *)
               IF DIR = HOR THEN
                    ENDCOORD↑.Y := STARTCOORD↑.Y
               ELSE
                    ENDCOORD↑.X := STARTCOORD↑.X;

          END (* THEN *);
          IF ((DIR = HOR) AND (STARTCOORD↑.Y <> ENDCOORD↑.Y) ) OR
             ((DIR = VERT) AND (STARTCOORD↑.X <> ENDCOORD↑.X) ) THEN
          Begin
               If Prevpath = Nil Then
                    ENDNOTSTRAIGHT(dir, prespath, nil, endcoord↑,
                                   startcoord, endcoord, Nil, Nil,
                                   box1, box2, num)
               Else
                    ENDNOTSTRAIGHT( DIR, PRESPATH , PREVPATH ,ENDCOORD↑ ,
                                    PRESPATH↑.STARTCOORD, PRESPATH↑.ENDCOORD,
                                    PREVPATH↑.STARTCOORD, PREVPATH↑.ENDCOORD,
                                    BOX1, BOX2,
                                    NUM )
          End;

     END (* WITH *)
END (* END PATH *);

    FUNCTION MOVEDIRLINE( COLST1: COORDLISTPTR;
                          BOX1, BOX2: BOXPTR;
                          VAR NUM: INTEGER): BOOLEAN;
    (* TO MOVE A DIRECT LINE FROM ITS PESENT POSITION *)
    VAR NEWEND: COORDPTR;
        RES: BOOLEAN;
    BEGIN
```

```
        RES := FALSE;
        WITH COLST1↑ DO
        BEGIN
             (* FIRST MARK THE PRESENT END AS TAKEN *)
             MARKCOORDTAKEN( COLST1,
                             ENDCOORD,
                             ENDEDGE,
                             BOX2 );
             (* NEXT GET ANOTHER END POINT *)
             NEWEND := GETRELEND( BOX2↑.RELSONEDGE,
                                  ENDEDGE,
                                  BOX2↑.CENTRE↑,
                                  BOX1↑.CENTRE↑ );
             (* NOW MARK PREV. COORD FREE AGAIN *)
             MARKCOORDTAKEN( FREE,
                             ENDCOORD,
                             ENDEDGE,
                             BOX2 );
             IF SPACEFOUND( NEWEND ) THEN
                  ENDCOORD := NEWEND
             ELSE IF PREVCOORD<>NIL THEN RES := TRUE
             ELSE BEGIN
                  (* CAN'T MOVE END, SO MOVE START *)
                  (* FIRST MARK THE PRESENT END AS TAKEN *)
                  MARKCOORDTAKEN( COLST1,
                                  STARTCOORD,
                                  STRTEDGE,
                                  BOX1 );
                  (* NEXT GET ANOTHER END POINT *)
                  NEWEND := GETRELEND( BOX1↑.RELSONEDGE,
                                       STRTEDGE,
                                       BOX1↑.CENTRE↑,
                                       BOX2↑.CENTRE↑ );
                  (* NOW MARK PREV. COORD FREE AGAIN *)
                  MARKCOORDTAKEN( FREE,
                                  STARTCOORD,
                                  STRTEDGE,
                                  BOX1 );
                  IF NOT SPACEFOUND( NEWEND ) THEN
                  BEGIN
                       (* CAN'T MOVE LINE, SO MUST LIVE WITH IT *)
                       NUM := NUM + 500; (* BAD SITUATION *)
                       RES := TRUE
                  END (* THEN *)
             END (* ELSE *)
        END (* WITH *);

        MOVEDIRLINE := RES
   END (* MOVEDIRLINE *);

        FUNCTION PATHHASNOREL( DIR: LINEDIRECTION;
                               COLST: COORDLISTPTR;
                               INC: INTEGER;
                               BOX1, BOX2: BOXPTR;
                               VAR NUM: INTEGER ): BOOLEAN;
```

```
VAR RES: BOOLEAN;
BEGIN
     RES := FALSE;
     WITH COLST↑ DO
     BEGIN
          IF same_coord(endcoord↑ , startcoord↑) THEN RES := TRUE
          ELSE IF NOT rel_in_same_path( COLST) THEN RES := TRUE
          ELSE BEGIN
               IF DIR = DIRT THEN RES := MOVEDIRLINE(COLST,BOX1,BOX2,NUM)
               ELSE IF PREVCOORD = NIL THEN
               BEGIN
                    res := true; num := -1;
               END (* PREVPATH = NIL *)
               ELSE BEGIN
                    (* SET APP. COORD TO THE NEXT VALUE TO TRY, UNTIL
                       A FREE LINE IS FOUND (IF AT ALL) *)
                    IF DIR = HOR THEN
                    BEGIN
                         STARTCOORD↑.Y := STARTCOORD↑.Y+INC;
                         PREVCOORD↑.ENDCOORD↑.Y := STARTCOORD↑.Y
                    END (* THEN *)
                    ELSE BEGIN
                         STARTCOORD↑.X := STARTCOORD↑.X+INC;
                         PREVCOORD↑.ENDCOORD↑.X := STARTCOORD↑.X
                    END (* ELSE *);
                    IF line_intersects_entity( PREVCOORD↑.LINEGRAD,
                                               PREVCOORD↑.LINECONST,
                                               PREVCOORD↑.STARTCOORD,
                                               PREVCOORD↑.ENDCOORD,
                                               BOX1,BOX2)        THEN
                    BEGIN
                         NUM := -1;
                         RES := TRUE
                    END;
               END (* ELSE *)
          END (* ELSE *)

     END (* WITH *);
     If Not res Then
        If num > (maxint div 2) Then num := maxint
                                Else num := num * 2; {penalise close lines}
     pathhasnorel := RES
END (* PATHHASNOREL *);

FUNCTION TRYDIRECTLINK( COORD1: COORDPTR;
                        STEDGE: RECTEDGE;
                        PREVPATH: COORDLISTPTR;
                        BOX: BOXPTR;
                        VAR NUM: INTEGER ): COORDLISTPTR;
(* TRY AND CONNECT POINTS DIRECTLY *)
VAR COLSTPT: COORDLISTPTR;
    OTHERCOORD: COORDPTR;
    M, C: REAL;
    EDGE: RECTEDGE;
BEGIN
```

```
        COLSTPT := NIL;
        EDGE := find_rect_region( BOX , COORD1↑ );
        OTHERCOORD := find_direct_link_end( coord1 , stedge , edge ,
                                            box1, box );
        IF SPACEFOUND(OTHERCOORD) THEN
        BEGIN
           (* SPACE ON EDGE , SO CAN TRY LINK *)
           M := grad_of_line( COORD1 , OTHERCOORD );
           C := find_const( M , COORD1 );
           (* SEE IF LINE FREE OF BOX INTERSECTION *)
           IF NOT line_intersects_entity( M,C,COORD1,OTHERCOORD,
                                          (*GLOBAL BOXES*)BOX1, BOX2 )
           THEN BEGIN
                (* LINE IS FREE, SO CAN DRAW IT *)
                NUM := NUM + 1;
                NEW( COLSTPT );
                WITH COLSTPT↑ DO
                BEGIN
                     NEXTCOORD := NIL;
                     PREVCOORD := PREVPATH;
                     STARTCOORD := COORD1;
                     ENDCOORD := OTHERCOORD;
                     LINEGRAD := M;
                     LINECONST := C;
                     STRORFIN := TRUE;
                     STRTEDGE := STEDGE;
                     ENDEDGE := EDGE
                END (* WITH *);
                (* ADD PENALTY FOR ANY INTERSECTING LINES *)
                NUM := NUM +
                       (no_of_intersecting_rels( COLSTPT)*INTRELPENALTY);
           END (* THEN *)
        END (* THEN *);
        TRYDIRECTLINK := COLSTPT
    END (* TRYDIRECTLINK *);

    FUNCTION PLOTRESTOFREL( COORD1: COORDPTR;
                            BOX: BOXPTR;
                            PREVDIR: LINEDIRECTION;
                            PREVPATH: COORDLISTPTR;
                            Var nooplinedrawn: Boolean;
                            VAR NUM: INTEGER ): COORDLISTPTR; FORWARD;

    PROCEDURE FINDHORPOS( COORD: COORDPTR;
                          OTHERCOMP: COMPONENT;
                          BOX1, BOX2: BOXPTR;
                          COLST: COORDLISTPTR );
    (* FIND HORIZONTAL POSN *)
    BEGIN
        WITH COLST↑ DO
        BEGIN
            NEW( ENDCOORD );
            LINEGRAD := 0;
            LINECONST := COORD↑.Y;
            ENDCOORD↑.X :=
```

```
                    FINDBESTPOS( HOR , COORD↑.X , OTHERCOMP ,
                                 COORD↑.Y, COORD, BOX1, BOX2);
          ENDCOORD↑.Y := COORD↑.Y
     END
END (* FINDHORPOS *);

PROCEDURE FINDVERTPOS( COORD: COORDPTR;
                       OTHERCOMP: COMPONENT;
                       BOX1, BOX2: BOXPTR;
                       COLST: COORDLISTPTR );
(* FIND VERTICAL POSN *)
BEGIN
     WITH COLST↑ DO
     BEGIN
          NEW( ENDCOORD );
          LINEGRAD := infinite_grad;
          LINECONST := COORD↑.X;
          ENDCOORD↑.Y :=
                    FINDBESTPOS( VERT , COORD↑.Y , OTHERCOMP,
                                 COORD↑.X, COORD, BOX1, BOX2);
          ENDCOORD↑.X := COORD↑.X
     END
END (* FINDVERTPOS *);

FUNCTION DRAWOPPLINE( COORD: COORDPTR;
                      BOX: BOXPTR;
                      STEDGE: RECTEDGE;
                      PREVDIR: LINEDIRECTION;
                      PREVPATH: COORDLISTPTR;
                      Var nooplinedrawn: Boolean;
                      VAR NUM: INTEGER ): COORDLISTPTR;
(* CAN'T DRAW LINE TOWARDS BOX., SO DRAW ONE AWAY *)
VAR CRDLST: COORDLISTPTR;
    DIR: LINEDIRECTION;
    PATHINC,TEMPNUM: INTEGER;

     PROCEDURE TRYVERTREVERSE( COMP: COMPONENT );
     (* TRY TO REVERSE IN VERTICAL  DIR. *)
     VAR INC: INTEGER;
     BEGIN
          DIR := VERT;
          INC := (HEIGHTOFENTITYBOX DIV 2) + 1;
          IF COMP < BOX↑.CENTRE↑.Y THEN INC := -INC
          ELSE IF (COMP = BOX↑.CENTRE↑.Y) AND (COMP < STRTBOX↑.CENTRE↑.Y)
               THEN INC := -INC;

          FINDVERTPOS( COORD ,(COMP+INC),NIL, NIL,CRDLST )
     END (* TRYVERTREVERSE *);

     PROCEDURE TRYHORREVERSE( COMP: COMPONENT );
     (* TRY TO REVERSE IN HOR. DIR. *)
     VAR INC: INTEGER;
     BEGIN
          DIR := HOR;
          INC := (WIDTHOFENTITYBOX DIV 2) + 1;
```

```
            IF COMP < BOX↑.CENTRE↑.X THEN INC := -INC
            ELSE IF (COMP = BOX↑.CENTRE↑.X) AND (COMP <= STRTBOX↑.CENTRE↑.X)
                 THEN INC := -INC;

            FINDHORPOS( COORD ,(COMP+INC),NIL, NIL,CRDLST ) -
       END (* TRYHORREVERSE *);

  BEGIN
       (* ARE ABOUT TO ADD NEW PATH *)
       (* IF IS START OF LINE, WANT TO ALLOW OTHER OP. LINES *)
       NOOPLINEDRAWN := (STEDGE <> NILEDGE);
       NUM := NUM + 1;
       NEW( CRDLST );
       If num < best_so_far Then crdlst := Nil
       Else
       WITH CRDLST↑ DO
       BEGIN
            STARTCOORD := COORD;
            endcoord := Nil;
            PREVCOORD := PREVPATH;
            NEXTCOORD := NIL;
            STRORFIN := (STEDGE <>NILEDGE);
            STRTEDGE := STEDGE;
            ENDEDGE := NILEDGE;
            IF PREVCOORD = NIL THEN PATHINC := 1
            ELSE WITH PREVCOORD↑ DO
                IF ((PREVDIR=VERT) AND (ENDCOORD↑.X>STARTCOORD↑.X)) OR
                   ((PREVDIR=HOR)AND (ENDCOORD↑.Y>STARTCOORD↑.Y)) THEN
                      PATHINC := 1
                      ELSE PATHINC := -1;
            REPEAT
                 IF ((OVERALLDIR <> PREVDIR) AND (OVERALLDIR = VERT)) OR
                    ((OVERALLDIR = PREVDIR) AND (OVERALLDIR = HOR)) THEN
                 BEGIN
                      TRYVERTREVERSE( STARTCOORD↑.Y );
                      IF same_coord(endcoord↑ , startcoord↑) THEN
                           TRYHORREVERSE( STARTCOORD↑.X )
                 END
                 ELSE BEGIN
                      TRYHORREVERSE( STARTCOORD↑.X );
                      IF same_coord(endcoord↑ , startcoord↑) THEN
                           TRYVERTREVERSE( STARTCOORD↑.Y )
                 END (* ELSE *);
            UNTIL PATHHASNOREL(DIR,CRDLST,PATHINC,STRTBOX,BOX,NUM)
                  Or (num >= best_so_far);

            IF (NUM <> -1) AND Not same_coord(ENDCOORD↑ , COORD↑) THEN
            BEGIN
                 (* HAS MOVED SO NEED TO FIND REST OF PATH *)
                 (* ADD LINE INTERSECTION PENALTY *)
                 NUM := NUM + (no_of_intersecting_rels( CRDLST ) *
                                   INTRELPENALTY);
                 If num < best_so_far Then
                      NEXTCOORD := PLOTRESTOFREL( ENDCOORD , BOX , DIR,
                                                  CRDLST,
```

```
                                                    nooplinedrawn,NUM )
                Else nextcoord := Nil
            END (* THEN *)
            ELSE BEGIN
                (* SAME POSITION AS STARTED AT, LINE NOT VIABLE *)
                NOOPLINEDRAWN := TRUE;
                NUM := MAXINT;
                CRDLST := NIL
            END (* ELSE *)
        END (* WITH *);

        DRAWOPPLINE := CRDLST
    END (* DRAWOPPLINE *);

    FUNCTION TRYPATH( DIR: LINEDIRECTION;
                      COORD: COORDPTR;
                      BOX1,
                      BOX: BOXPTR;
                      STEDGE: RECTEDGE;
                      PREVPATH: COORDLISTPTR;
                      Var nooplinedrawn: Boolean;
                      VAR NUM: INTEGER ): COORDLISTPTR;
    (* FIND PATH FROM COORD TO BOX *)
    VAR FIRSTPATH: COORDLISTPTR;
        CENT: COORDINATE;
        PATHINC: INTEGER;
    BEGIN
        CENT := BOX↑.CENTRE↑;
        (* ARE ABOUT TO ADD NEW PATH *)
        NUM := NUM + 1;
        NEW( FIRSTPATH );
        If num > best_so_far Then firstpath := Nil
        Else
        WITH FIRSTPATH↑ DO
        BEGIN
            STARTCOORD := COORD;
            endcoord := Nil;
            PREVCOORD := PREVPATH;
            NEXTCOORD := NIL;
            STRORFIN := (STEDGE <> NILEDGE);
            STRTEDGE := STEDGE;
            ENDEDGE := NILEDGE;

            IF PREVCOORD = NIL THEN PATHINC := 1
            ELSE WITH PREVCOORD↑ DO
                IF ((DIR=HOR) AND (ENDCOORD↑.X>STARTCOORD↑.X)) OR
                   ((DIR=VERT)  AND (ENDCOORD↑.Y>STARTCOORD↑.Y)) THEN
                    PATHINC := 1
                    ELSE PATHINC := -1;
            REPEAT
                IF DIR = HOR THEN FINDHORPOS( STARTCOORD , CENT.X ,
                                              BOX1, BOX,
                                              FIRSTPATH )
                             ELSE FINDVERTPOS( STARTCOORD, CENT.Y ,
                                               BOX1, BOX,
```

```
                                                    FIRSTPATH );
            UNTIL PATHHASNOREL(DIR,FIRSTPATH,PATHINC,BOX1,BOX,NUM)
                  Or (num >= best_so_far);

            IF NUM <> -1 THEN
                 NUM := NUM + (no_of_intersecting_rels( FIRSTPATH ) *
                               INTRELPENALTY );
            (* SEE IF HAVE ACTUALLY REACHED BOX *)
            IF (NUM <> -1) AND
               ((ENDCOORD^.X <= BOX^.XCOORDS[GTXBND]) AND
                (ENDCOORD^.X >= BOX^.XCOORDS[LESSXBND]) AND
                (ENDCOORD^.Y <= BOX^.YCOORDS[GTYBND]) AND
                (ENDCOORD^.Y >= BOX^.YCOORDS[LESSYBND])) THEN
            Begin
                 (* HAVE REACHED BOX, SO END LIST *)
                 If num <= best_so_far Then
                      ENDPATH( DIR , FIRSTPATH , PREVPATH, BOX, NUM)
                 Else nextcoord := Nil
            End
            ELSE IF (NUM <> -1) AND Not same_coord(endcoord^ ,startcoord^)
            THEN Begin
                 (* HAS MOVED SO NEED TO FIND REST OF PATH *)
                 If num < best_so_far Then
                      NEXTCOORD := PLOTRESTOFREL( ENDCOORD , BOX , DIR,
                                                  FIRSTPATH, nooplinedrawn,NUM )
                 Else nextcoord := Nil
            End
            ELSE BEGIN
                 (* SAME POSITION AS STARTED AT, LINE NOT VIABLE *)
                 NUM := MAXINT;
                 If firstpath <> Nil Then Dispose( firstpath );
                 FIRSTPATH := NIL
            END (* ELSE *)

       END (* WITH *);
       TRYPATH := FIRSTPATH
  END (* TRYPATH *);

  FUNCTION PLOTRESTOFREL(* COORD1: COORDPTR;
                           BOX: BOXPTR;
                           PREVDIR: LINEDIRECTION;
                           PREVPATH: COORDLISTPTR;
                           Var nooplinedrawn: Boolean;
                           VAR NUM: INTEGER): COORDLISTPTR *);
  (* PLOT REST OF A RELATION FROM COORD1 TO BOX *)
  VAR VERTCOORDLST, HORCOORDLST, CRDLST: COORDLISTPTR;
      NOHOR, NOVERT: INTEGER;
  BEGIN
       CRDLST := NIL;
       vertcoordlst := Nil;
       horcoordlst := Nil;
       NOVERT := num;
       IF PREVDIR <> VERT THEN
       Begin
            VERTCOORDLST := TRYPATH(VERT , COORD1 ,NIL,
```

```
                                        BOX,NILEDGE,
                                        PREVPATH, nooplinedrawn, NOVERT );
          If novert < best_so_far Then best_so_far := novert
     End;

     NOHOR := num;
     IF PREVDIR <>HOR THEN
     Begin
          HORCOORDLST := TRYPATH( HOR , COORD1 , NIL,
                                        BOX ,NILEDGE,
                                        PREVPATH, nooplinedrawn, NOHOR );
          If nohor< best_so_far Then best_so_far := nohor
     End;

     (* COMPARE TWO PATHS TO SEE WHICH IS BEST *)
     IF ((NOVERT = MAXINT) OR (NOVERT = num)) AND
        ((NOHOR = MAXINT)  OR (NOHOR = num) ) THEN
     BEGIN
          (* SERIOUS POSITION, LINE DIFFICULT *)
          NUM := NUM + 2;
          If num > best_so_far Then crdlst := Nil
          Else Begin
               (* FIRST TRY DIRECT LINE *)
               (* ONLY WANT TO ALLOW ONE OPP. LINE *)
               IF NOOPLINEDRAWN THEN
                    CRDLST := DRAWOPPLINE( COORD1 , BOX , NILEDGE,
                                        PREVDIR,PREVPATH,nooplinedrawn,NUM);
               IF CRDLST = NIL THEN
                    NUM := MAXINT
          End
     END (* THEN *)
     ELSE IF (NOHOR = num) OR
             ((NOVERT< >num) AND (NOVERT <= NOHOR)) THEN
     BEGIN
          (* VERTICAL PATH IS BEST *)
          CRDLST := VERTCOORDLST;
          dispose_list( horcoordlst );
          NUM := NOVERT
     END
     ELSE BEGIN
          (* HORIZONTAL PATH IS BEST *)
          CRDLST := HORCOORDLST;
          dispose_list( vertcoordlst );
          NUM := NOHOR
     END (* ELSE *);
     PLOTRESTOFREL := CRDLST
END (* PLOTRESTOFREL *);

Procedure removeendrelcross( edge: rectedge;
                             edgerel: edgerelstarts;
                             inrel: coordlistptr;
                             box, othbox: boxptr ); Forward;

FUNCTION swap_line_ends_is_ok(REL1, REL2: COORDLISTPTR;
                         EDGE: RECTEDGE;
```

```
                    BOX, OTHBOX: BOXPTR;
                    VAR REL2BOX: BOXPTR ): BOOLEAN;
(* TO PERFORM SWAPPING OF LINE ENDS *)
VAR T1COLST, T2COLST: COORDLST;
    ISREL1END, ISREL2END: BOOLEAN;
    ok: BOOLEAN;
    NUM: INTEGER;
    t2ptr : coordlistptr;

    PROCEDURE SWAP( REL1CO, REL2CO: COORDPTR );
    VAR TEMPCO: COORDINATE;
    BEGIN
        TEMPCO := REL2CO^;
        REL2CO^ := REL1CO^;
        REL1CO^ := TEMPCO;
        MARKCOORDTAKEN( REL1, REL1CO, EDGE, BOX );
        MARKCOORDTAKEN( REL2, REL2CO, EDGE, BOX );
    END (* SWAP *);

    PROCEDURE RESTORE;
    BEGIN
        (* LINE NOT POSSIBLE *)
        REL1^ := T1COLST;
        REL2^ := T2COLST;
        ok := FALSE;
        IF ISREL1END THEN
        BEGIN
            MARKCOORDTAKEN( REL1, REL1^.ENDCOORD,
                            EDGE, BOX );
            IF ISREL2END THEN
                MARKCOORDTAKEN( REL2, REL2^.ENDCOORD,
                                EDGE, BOX )
            ELSE
                MARKCOORDTAKEN( REL2, REL2^.STARTCOORD,
                                EDGE, BOX )
        END (* THEN *)
        ELSE BEGIN
            MARKCOORDTAKEN( REL1, REL1^.STARTCOORD,
                            EDGE, BOX );
            IF ISREL2END THEN
                MARKCOORDTAKEN( REL2, REL2^.ENDCOORD,
                                EDGE, BOX )
            ELSE
                MARKCOORDTAKEN( REL2, REL2^.STARTCOORD,
                                EDGE, BOX )
        END (* ELSE *)
    END (* RESTORE *);

    PROCEDURE TRYLINE(REL: COORDLISTPTR;
                      ISRELEND: BOOLEAN;
                      BOXB: BOXPTR );
    VAR DIR: LINEDIRECTION;

        Procedure check_direct_line;
        { To ensure a reasonable change of a direct line }
```

```
Var change_made: Boolean;
    othercoord, coord, oldco: coordptr;
    other_edge: rectedge;
    comp: component;
Begin
     change_made := False;
     With rel↑ Do
     Begin
          If (linegrad = 0) Or (linegrad = infinite_grad) Then
          Begin
               { Must have been vertical or horizontal, so
                 try to keep as such }
               If isrelend Then
               Begin
                    othercoord := startcoord;
                    coord := endcoord;
                    other_edge := strtedge
               End
               Else Begin
                    othercoord := endcoord;
                    coord := startcoord;
                    other_edge := endedge
               End;
               If linegrad = 0 Then comp := coord↑.y
                               Else comp := coord↑.x;
               If posn_is_free( comp , other_edge , boxb ) Then
               Begin
                    New( oldco );
                    oldco↑ := othercoord↑;
                    change_made := True;
                    If linegrad = 0  Then othercoord↑.y := comp
                                     Else othercoord↑.x := comp;
                    If line_intersects_entity( linegrad ,
                                               lineconst ,
                                               coord ,
                                               othercoord ,
                                               box,boxb)
                    Then Begin
                         { Not possible, so forget }
                         othercoord↑ := oldco↑;
                         change_made := False
                    End Else Begin
                         markcoordtaken(rel,othercoord,
                                        other_edge,boxb);
                         markcoordtaken(free,oldco,
                                        other_edge,boxb);
                         removeendrelcross( other_edge ,
                          boxb↑.relsonedge, rel ,
                          boxb , box )
                    End {Else };
                    Dispose( oldco )
               End
          End {then};
          If Not change_made Then
          Begin
```

```
                              LINEGRAD :=grad_of_line( STARTCOORD , ENDCOORD );
                              LINECONST := find_const( LINEGRAD , STARTCOORD );
                              IF line_intersects_entity( LINEGRAD, LINECONST,
                                                         STARTCOORD,ENDCOORD,
                                                         BOX, BOXB )
                              THEN RESTORE
                         End
                    End {With}
               End {check_direct_line};
BEGIN
               WITH REL^ DO
               BEGIN
                    IF ((PREVCOORD = NIL) AND (NEXTCOORD = NIL)) OR
                       ((LINEGRAD<>0) AND (LINEGRAD<>infinite_grad))  THEN
                         check_direct_line
                    ELSE BEGIN
                         IF LINEGRAD = 0 THEN DIR := HOR
                                         ELSE DIR := VERT;
                         NUM := 0;
                         IF ISRELEND THEN
                              ENDNOTSTRAIGHT(DIR,REL,PREVCOORD,ENDCOORD^,
                                             STARTCOORD, ENDCOORD,
                                             PREVCOORD^.STARTCOORD,
                                             PREVCOORD^.ENDCOORD,
                                             BOX, BOXB, NUM )
                         ELSE ENDNOTSTRAIGHT(DIR,REL,NEXTCOORD,STARTCOORD^,
                                             ENDCOORD, STARTCOORD,
                                             NEXTCOORD^.ENDCOORD,
                                             NEXTCOORD^.STARTCOORD,
                                             BOX, BOXB, NUM );
                         IF ((LINEGRAD=0) AND (STARTCOORD^.Y <> ENDCOORD^.Y))
                          OR ((LINEGRAD=infinite_grad)
                               AND (STARTCOORD^.X <> ENDCOORD^.X))
                          THEN RESTORE
                    END (* ELSE *)
               END (* WITH *);

          END (* TRYLINE *);

     BEGIN
          ok := TRUE;
          T1COLST := REL1^;
          T2COLST := REL2^;

          ISREL1END := (REL1^.ENDEDGE = EDGE );
          ISREL2END := (REL2^.ENDEDGE = EDGE );
          (* SWAP LINE AROUND *)
          IF ISREL1END THEN
          BEGIN
               IF ISREL2END THEN SWAP( REL1^.ENDCOORD, REL2^.ENDCOORD)
                            ELSE SWAP( REL1^.ENDCOORD,REL2^.STARTCOORD)
          END
          ELSE IF ISREL2END THEN SWAP(REL1^.STARTCOORD, REL2^.ENDCOORD)
                            ELSE SWAP(REL1^.STARTCOORD,REL2^.STARTCOORD);
```

```
          (* ENSURE IS VALID TO DO SO & RESET VALUES IF ISN'T,
             SET UP REST OF VALUIES IF IS *)

          TRYLINE( REL1 , ISREL1END, OTHBOX  );(* ok IS SET IF LINE NOT POSS *)
          IF ok THEN
          Begin
               t2ptr := rel2;
               If isrel2end Then
               Begin
                    While t2ptr↑.prevcoord <> Nil Do
                         t2ptr := t2ptr↑.prevcoord;
                    rel2box := findboxaroundcoord( t2ptr↑.startcoord )
               End
               Else Begin
                    While t2ptr↑.nextcoord <> Nil Do
                         t2ptr := t2ptr↑.nextcoord;
                    rel2box := findboxaroundcoord( t2ptr↑.endcoord )
               End;
               tryline( rel2 , isrel2end , rel2box )
          End;
          swap_line_ends_is_ok :=  ok
     END (* swap_line_ends_is_ok *);

     PROCEDURE REMOVEENDRELCROSS(* EDGE: RECTEDGE;
                                   EDGEREL: EDGERELSTARTS;
                                   INREL: COORDLISTPTR;
                                   box,
                                   othBOX: BOXPTR *);
(* TO ATTEMPT TO REMOVE ANY CROSSING OF END RELS *)
VAR MAXNORELS: INTEGER;
    REL2BOX: BOXPTR;

     PROCEDURE CHECKNOENDRELCROSS;
     (* TO REMOVE CROSS IF ONE EXISTS *)
     VAR I: INTEGER;
         oldrel2,oldrel,REL,REL2: COORDLISTPTR;
     BEGIN
          I := 1;
          REL := INREL;
          While rel↑.prevcoord <> Nil Do rel := rel↑.prevcoord;
          WHILE (I <= MAXNORELS) DO
          BEGIN
               (* FIND START OF LINE *)
               IF (EDGEREL[EDGE,I]<>REL) AND (EDGEREL[EDGE,I] <> NIL)
                  And (edgerel[edge,i] <> inrel) THEN
               BEGIN
                    REL2 := EDGEREL[EDGE,I];
                    oldrel2 := rel2;
                    WHILE REL2↑.PREVCOORD <> NIL DO
                         REL2 := REL2↑.PREVCOORD;
                    IF line_and_line_intersection( REL , REL2 ) THEN
                    BEGIN
                         IF swap_line_ends_is_ok( inREL,EDGEREL[EDGE,I], EDGE,
                                        BOX, OTHBOX, REL2BOX  ) THEN
                         BEGIN
```

```
                                inREL := oldREL2;
                                rel := inrel;
                                While rel↑.prevcoord<> Nil Do
                                     rel := rel↑.prevcoord;
                                OTHBOX := REL2BOX
                           END
                      END
                 END (* THEN *);

                 I := I + 1
            END (* WHILE *)
       END (* CHECKNOENDRELCROSS *);

BEGIN
            IF (EDGE = TOP) OR (EDGE = BOTTOM) THEN
            MAXNORELS := MAXNOXRELS
       ELSE
            MAXNORELS := MAXNOYRELS;
       CHECKNOENDRELCROSS;

END (* REMOVEENDRELCROSS *);

       PROCEDURE MARKENDSOFPATH( COLSTPT: COORDLISTPTR );
       (* TO MARK EACH END OF THE LINE DENOTED BY CO. LIST AS START & FINISH *)
       VAR CLPT: COORDLISTPTR;
           POSN: INTEGER;
           ENDCOORD: COORDPTR;
       BEGIN
            CLPT := COLSTPT;
            IF CLPT<> NIL THEN
            BEGIN
                 (* MARK START OF PATH *)
                 MARKCOORDTAKEN( CLPT,
                                 CLPT↑.STARTCOORD,
                                 CLPT↑.STRTEDGE,
                                 BOX1 );

                 REMOVEENDRELCROSS( CLPT↑.STRTEDGE,
                                    BOX1↑.RELSONEDGE ,
                                    CLPT,
                                    BOX1,box2);

                 (* FIND AND OF PATH *)
                 WHILE CLPT↑.NEXTCOORD<  >NIL DO
                      CLPT := CLPT↑.NEXTCOORD;
                 (* END OF PATH AT ENTRY FOUND *)

                 MARKCOORDTAKEN( CLPT,
                                 CLPT↑.ENDCOORD,
                                 CLPT↑.ENDEDGE,
                                 BOX2 );

                 REMOVEENDRELCROSS( CLPT↑.ENDEDGE,
                                    BOX2↑.RELSONEDGE ,
                                    CLPT,
```

```
                                   BOX2,box1);
     END (* IF *);

END (* MARKENDSOFPATH *);

FUNCTION FINDSTART( BOX1 , BOX2: BOXPTR;
                    DIROFLINE: LINEDIRECTION;
                    VAR STEDGE: RECTEDGE;
                    Var crdstrt2: coordptr ): COORDPTR;
(* TO FIND A START COORD ON EDGE OF BOX1 IN DIRECTION OF BOX2 *)
VAR CENTRE1, CENTRE2: COORDINATE;
    STARTCOORD: COORDPTR;
BEGIN
     startcoord := Nil;
     crdstrt2 := Nil;
     (* FIRST FIND DIRECTION OF BOX2 FROM BOX1 *)
     CENTRE1 := BOX1^.CENTRE;
     CENTRE2 := BOX2^.CENTRE;

     (* NEED TO SEE IF A DIRECT LINE IS HORIZONTAL OR VERTICAL &
        CHANGE DIROFLINE IF SO TO INDICATE THIS (SIMPLER PROCESSING
        IN THIS CASE) *)
     IF DIROFLINE = DIRT THEN
     BEGIN
          IF CENTRE2.X = CENTRE1.X THEN DIROFLINE := VERT
          ELSE IF CENTRE2.Y = CENTRE1.Y THEN DIROFLINE := HOR
     END (* IF *);

     (* DEAL WITH DIR OF LINE, EDGE  DEPENDS ON DIRECTION *)
     CASE DIROFLINE OF
          VERT: (* CAN BE ABOVE OR BELOW *)
               IF (CENTRE2.Y > CENTRE1.Y) THEN
                    STEDGE := TOP
               ELSE If (centre2.y < centre1.y) Then
                    STEDGE := BOTTOM
               Else
                    stedge := niledge;
          HOR: (* CAN BE ON EITHER SIDE *)
               IF (CENTRE2.X > CENTRE1.X) THEN
                    STEDGE := RIGHT
               ELSE If (centre2.x < centre1.x) Then
                    STEDGE := LEFT
               Else
                    stedge := niledge;
          DIRT: (* CAN BE IN ONE OF FOUR GENERAL DIRECTIONS *)
               STEDGE := find_rect_region( BOX1 , CENTRE2 )
     END (* CASE *);

     If stedge <> Niledge Then
          STARTCOORD := GETRELEND( BOX1^.RELSONEDGE, STEDGE,
                                   CENTRE2 , CENTRE1 )
     Else If dirofline = vert Then
     Begin
          { Need to start from 2 edges, so 2 starts }
          STARTCOORD := GETRELEND( BOX1^.RELSONEDGE, top,
```

```
                                        CENTRE2 , CENTRE1 );
            crdstrt2 := GETRELEND( BOX1^.RELSONEDGE, bottom,
                                        CENTRE2 , CENTRE1 );
            If Not spacefound( crdstrt2 ) Then
            { 2nd start not viable }
                crdstrt2 := Nil
        End
        Else If dirofline = hor Then
        Begin
            { Need to start from 2 edges, so 2 starts }
            STARTCOORD := GETRELEND( BOX1^.RELSONEDGE, left,
                                        CENTRE2 , CENTRE1 );
            crdstrt2 := GETRELEND( BOX1^.RELSONEDGE, right,
                                        CENTRE2 , CENTRE1 );
            If Not spacefound( crdstrt2 ) Then
            { 2nd start not viable }
                crdstrt2 := Nil
        End;
        IF NOT SPACEFOUND( STARTCOORD ) THEN
            (* start NOT viable *)
            STARTCOORD := NIL;
        FINDSTART := STARTCOORD

    END (* FINDSTART *);

    FUNCTION SHOULDDRAWOPLINE( DIR: LINEDIRECTION;
                               BOX1, BOX2: BOXPTR ): BOOLEAN;
    (* TO SEE IF OPP. LINE SHOULD BE DRAWN *)
    BEGIN
        SHOULDDRAWOPLINE :=
            ( (DIR = HOR) AND
              (((BOX2^.CENTRE^.X >= BOX1^.XCOORDS[LESSXBND]) AND
                (BOX2^.CENTRE^.X <= BOX1^.CENTRE^.X))   OR
               ((BOX2^.CENTRE^.X <  BOX1^.XCOORDS[GTXBND]) AND
                (BOX2^.CENTRE^.X >= BOX1^.CENTRE^.X))   )       ) OR
            ( (DIR = VERT) AND
              (((BOX2^.CENTRE^.Y >= BOX1^.YCOORDS[LESSYBND]) AND
                (BOX2^.CENTRE^.Y <= BOX1^.CENTRE^.Y))   OR
               ((BOX2^.CENTRE^.Y <  BOX1^.YCOORDS[GTYBND]) AND
                (BOX2^.CENTRE^.Y >= BOX1^.CENTRE^.Y))   )       )
        END (* SHOULDDRAWOPLINE *);

    Function bad_rel( n1, n2 : Integer ): Boolean;
    { To see if an attempted path was viable }
    Begin
        bad_rel := (
            ((n1 = Maxint) And (n2 = Maxint))   Or
            ((n1 = Maxint) And (n2 = 0))        Or
            ((n1 = 0) And (n2 = Maxint))
                )
    End {bad-rel};

    FUNCTION BEGINPATH( DIR: LINEDIRECTION;
                        BOX1, BOX2: BOXPTR;
                        VAR NUM: INTEGER ): COORDLISTPTR;
```

```
(* TO PATH IN SPECIFIED DIRECTION *)
VAR CRDSTART, crdstrt2: COORDPTR;
    CRDLST, crdlst2: COORDLISTPTR;
    STEDGE: RECTEDGE;
    num1, num2: Integer;
BEGIN
     NUM := 0; num1 := 0; num2 := 0;
     OVERALLDIR := DIR;
     CRDLST := NIL; crdlst2:= Nil;
     (* FIND START OF LINE *)
     CRDSTART := FINDSTART( BOX1 , BOX2 , DIR , STEDGE , crdstrt2 );
     IF CRDSTART <> NIL THEN
     BEGIN
          STRTBOX := BOX1;
          NOOPLINEDRAWN := TRUE;
          SECONDTRY := FALSE;
          IF (stedge <> niledge) And
             (Not SHOULDDRAWOPLINE( DIR , BOX1 , BOX2 )) THEN
                CRDLST := TRYPATH( DIR , CRDSTART,BOX1,BOX2,
                                   STEDGE, NIL, nooplinedrawn,NUM)
          Else BEGIN
               { need to draw op. line
                 if stedge is niledge then is on direct line
                 so need to try both start sides for direction }
               If stedge <> niledge Then
               Begin
                    { just try start edge have }
                    IF DIR = HOR THEN
                         CRDLST := DRAWOPPLINE( CRDSTART,BOX2,STEDGE,
                                        VERT, NIL, nooplinedrawn,NUM )
                    ELSE
                         CRDLST := DRAWOPPLINE( CRDSTART,BOX2,STEDGE,
                                        HOR, NIL, nooplinedrawn,NUM )
               End Else Begin
                    { try both start edges }
                    If dir = hor Then
                    Begin
                        CRDLST := DRAWOPPLINE( CRDSTART,BOX2,left,
                                        VERT, NIL, nooplinedrawn,NUM1 );
                        If crdstrt2 <> Nil Then
                        CRDLST2 := DRAWOPPLINE( CRDSTRT2,BOX2,right,
                                        vert, NIL, nooplinedrawn,NUM2 )
                    End Else Begin
                        CRDLST := DRAWOPPLINE( CRDSTART,BOX2,top,
                                        hor, NIL, nooplinedrawn,NUM1 );
                        If crdstrt2 <> Nil Then
                        CRDLST2 := DRAWOPPLINE( CRDSTRT2,BOX2,bottom,
                                        hor, NIL, nooplinedrawn,NUM2 )
                    End;
                    If (num1 = 0) And (num2 = 0) Then num := 0
                    Else If bad_rel( num1 , num2) Then
                    Begin
                         num := Maxint;
                         dispose_list(crdlst2)
                    End
```

```
                    Else If (num2 = 0) Or ((num1 <> 0) And (num1 < num2))
                    Then Begin
                         num := num1;
                         dispose_list( crdlst2 )
                    End
                    Else Begin
                         num := num2;
                         dispose_list( crdlst );
                         crdlst := crdlst2
                    End
               End
          END;

     END;

     BEGINPATH := CRDLST
END (* BEGINPATH *);

FUNCTION REVERSELIST( CRDLST: COORDLISTPTR ): COORDLISTPTR;
(* TO REVERSE LIST INPUT *)
VAR TEMPPT: COORDPTR;
    TEMPEDGE: RECTEDGE;
BEGIN
     IF CRDLST↑.NEXTCOORD <> NIL THEN
     WITH CRDLST↑ DO
     BEGIN
          TEMPEDGE := ENDEDGE;
          ENDEDGE := STRTEDGE;
          STRTEDGE := TEMPEDGE
     END;
     WHILE (CRDLST↑.NEXTCOORD <>NIL) DO
     BEGIN
          CRDLST := CRDLST↑.NEXTCOORD;
          WITH CRDLST↑.PREVCOORD↑ DO
          BEGIN
               NEXTCOORD := PREVCOORD;
               PREVCOORD := CRDLST;
               TEMPPT := ENDCOORD;
               ENDCOORD := STARTCOORD;
               STARTCOORD := TEMPPT;
          END (* WITH *);
     END (* WHILE *);
     WITH CRDLST↑ DO
     BEGIN
          TEMPEDGE := ENDEDGE;
          ENDEDGE := STRTEDGE;
          STRTEDGE := TEMPEDGE;
          NEXTCOORD := PREVCOORD;
          PREVCOORD := NIL;
          TEMPPT := ENDCOORD;
          ENDCOORD := STARTCOORD;
          STARTCOORD := TEMPPT;
     END (* WITH *);
     REVERSELIST := CRDLST
END (* REVERSELIST *);
```

```
FUNCTION TRYBOTHWAYS( DIR: LINEDIRECTION;
                      VAR NUM: INTEGER ): COORDLISTPTR;
(* TO TRY FROM BOTH SIDES *)
VAR NO12, NO21: INTEGER;
    CRDLST12, CRDLST21, CRDLST: COORDLISTPTR;
BEGIN
     CRDLST := NIL;
     num := 0;
     crdlst12 := Nil;
     crdlst21 := Nil;
     NO12 := 0;
     CRDLST12 := BEGINPATH( DIR , BOX1 , BOX2, NO12 );
     IF NO12 = 1 THEN
     BEGIN
          (* DIRECT LINE WITH NO INTERSECTIONS, CAN'T DO BETTER *)
          NUM := NO12;
          CRDLST := CRDLST12
     END
     ELSE BEGIN
          NO21 := 0;
          CRDLST21 := BEGINPATH( DIR , BOX2 , BOX1 , NO21 );
          IF (NO21 = 0) AND (NO12 = 0) THEN NUM := 0
          ELSE IF bad_rel( no12 , no21 ) THEN
          BEGIN
               NUM := MAXINT;
               dispose_list(crdlst12);
               dispose_list(crdlst21);
          END
          ELSE IF (NO21 = 0) OR ((NO12 <> 0) AND (NO12 <= NO21)) THEN
          BEGIN
               NUM := NO12;
               crdlst := crdlst12;
               dispose_list( crdlst21 );
          END
          ELSE BEGIN
               NUM := NO21;
               CRDLST := REVERSELIST( CRDLST21 );
               dispose_list( crdlst12 )
          END (* ELSE *)
     END (* ELSE *);
     TRYBOTHWAYS := CRDLST
END (* TRYBOTHWAYS *);

FUNCTION INVOLREL( CRDSTART: COORDPTR;
                   STEDGE: RECTEDGE ): COORDLISTPTR;
VAR CRDLST: COORDLISTPTR;
    NUM: INTEGER;
BEGIN
     OVERALLDIR := HOR;
     { stedge likely to come back nil, so probably won't work }
     CRDSTART := FINDSTART( BOX1 , BOX2 , VERT , STEDGE , crdstart2 );
     NUM := 0;
     CRDLST := NIL;
     IF CRDSTART <> NIL THEN
```

```
          BEGIN
               (* MARK COORD TAKEN TO ENSURE DOESN'T ARRIVE AT SAME PLACE*)
               NEW( CRDLST );
               MARKCOORDTAKEN( CRDLST , CRDSTART , top, BOX1 );
               NOOPLINEDRAWN := TRUE;
               SECONDTRY := FALSE;
               CRDLST := DRAWOPPLINE( CRDSTART , BOX2 , top, HOR, NIL,
                                      nooplinedrawn, NUM );
          END (* THEN *);
          INVOLREL := CRDLST
     END (* INVOLREL *);


BEGIN
     (* TRY A DIRECT LINE BETWEEN ENTITIES *)
     (* FIRST FIND START OF LINE *)
     CRDLST := NIL;
     NODIR := 0;
     DIRCRDSTART := FINDSTART( BOX1 , BOX2 , DIRT , STEDGE , crdstart2);
     IF DIRCRDSTART <> NIL THEN
     BEGIN
          IF BOX1 = BOX2 THEN
               CRDLST := INVOLREL( DIRCRDSTART , STEDGE )
          ELSE
               CRDLST := TRYDIRECTLINK( DIRCRDSTART , STEDGE ,
                                        NIL, BOX2, NODIR );
     END (* THEN *);
     IF (CRDLST = NIL) OR (NODIR <> 1)  THEN
     BEGIN
          best_so_far := Maxint;
          VERTCOORDLST := TRYBOTHWAYS( VERT  ,  NOVERT);
          HORCOORDLST := TRYBOTHWAYS( HOR ,  NOHOR );
     (* COMPARE PATHS TO SEE WHICH IS BEST *)
          IF (NOHOR = 0) AND (NOVERT = 0) THEN
          BEGIN
               IF NODIR = 0 THEN (* NOT EVEN DIRECT LINE VIABLE *)
                    WRITELN('ERROR - TOO MANY RELS. CANNOT FIND A START POSN');
               dispose_list( horcoordlst );
               dispose_list( vertcoordlst );
          END
          ELSE IF bad_rel( nohor , novert ) THEN
          BEGIN
               IF NODIR = 0 THEN
                    WRITELN('ERROR - LINE IS NOT VIABLE');
               dispose_list( horcoordlst );
               dispose_list( vertcoordlst );
          END (* THEN *)
          (*ELSE IF (NODIR <>0) AND
                  ((NOHOR = 0) OR (NODIR <= NOHOR)) AND
                  ((NOVERT = 0) OR (NODIR <= NOVERT)) THEN
               (* KEEP DIRECT LINE, INTERSECTIONS AND ALL
               dispose_list( vertcoordlst );
               dispose_list( horcoordlst )
          END*)
          ELSE IF (NOHOR = 0) OR ((NOVERT <> 0) AND (NOVERT <= NOHOR)) THEN
```

```
          BEGIN
               (* VERTICAL PATH IS BEST *)
     dispose_list( crdlst );
               CRDLST := VERTCOORDLST;
               (* GET RID OF OTHER LIST IF IT EXISTS *)
               dispose_list( horcoordlst );
          END
          ELSE BEGIN
               (* HORIZONTAL PATH IS BEST *)
       dispose_list( crdlst );
               CRDLST := HORCOORDLST;
               dispose_list( vertcoordlst );
          END (* ELSE *);
     END (* END IF COORDLIST *);
     (* MARK START & ENDS OF PATH ON ENTITY BOXES *)
     IF CRDLST    NIL THEN MARKENDSOFPATH( CRDLST );
     PLOTRELATIONSHIP := CRDLST
END (* PLOTRELATIONSHIP *);


       PROCEDURE SETBOXAT(BOX: BOXPTR;
                          X,Y: COMPONENT );
     (* TO SET BOX POSN AT GIVEN COORD *)
     BEGIN
          WITH BOX↑ DO
          BEGIN

               CENTRE↑.X := X;
               CENTRE↑.Y := Y;

               (* SET UP VERTICES COORDS *)
               XCOORDS[1] := X - HALFXGAP;
               YCOORDS[1] := Y + HALFYGAP;
               XCOORDS[2] := X + HALFXGAP;
               YCOORDS[2] := Y + HALFYGAP;
               XCOORDS[3] := X + HALFXGAP;
               YCOORDS[3] := Y - HALFYGAP;
               XCOORDS[4] := X - HALFXGAP;
               YCOORDS[4] := Y - HALFYGAP
           END (* WITH *);

     END (* SETBOXAT *);

     FUNCTION SETUPBOX( COORD: COORDINATE ): BOXPTR;
     (* TO SET UP & INTIALISE A BOX ENTRY *)
     VAR BOX: BOXPTR;
         I: INTEGER;
     BEGIN
          NEW( BOX );
          WITH BOX↑ DO
          BEGIN
               NEW( CENTRE );

               SETBOXAT( BOX , COORD.X , COORD.Y );
               (* SET UP RELSONEDGE TO FREE *)
```

```
            FOR I := 1 TO MAXNOXRELS DO
            BEGIN
                 RELSONEDGE[ TOP , I ] := FREE;
                 RELSONEDGE[ BOTTOM , I ] := FREE
            END (* FOR *);
            FOR I := 1 TO MAXNOYRELS DO
            BEGIN
                 RELSONEDGE[ RIGHT , I ] := FREE;
                 RELSONEDGE[ LEFT , I ] := FREE
            END (* FOR *)
       END (* WITH BOX↑ *);

       SETUPBOX := BOX
  END (* SETUPBOX *);

  FUNCTION TRYCARDINALPTS( INCOORD:COORDINATE;
                           XINC, YINC: INTEGER;
                           VAR BOX: BOXPTR ): BOOLEAN;
  (* TO TRY HOR. & VERT LINES FIRST *)
  VAR RES: BOOLEAN;
      COORD: COORDINATE;
  BEGIN
       (* BOX IS ALREADY SET FOR RIGHT POSN *)
       res := False;
       COORD := INCOORD;
       WITH COORD DO
       IF LASTBOXBELOW THEN
            X := X + XINC
       ELSE Y := Y - YINC;
       If average_rel  And ((incoord.x  <>orign↑.x) Or
                            (incoord.y  <>orign↑.y) ) Then
       Begin
            box := setupbox( incoord );
            If free_position( box ) Then res := True
            Else setboxat( box , coord.x , coord.y )
       End Else
            box := setupbox( coord );
       If Not res Then
       Begin
            IF free_position( BOX ) THEN RES := TRUE
            ELSE BEGIN
                 IF LASTBOXBELOW THEN
                      SETBOXAT( BOX , INCOORD.X , (INCOORD.Y-YINC))
                 ELSE
                      SETBOXAT( BOX , (INCOORD.X+XINC) , INCOORD.Y );
                 RES := free_position( BOX )
            END (* ELSE *);
            LASTBOXBELOW := NOT LASTBOXBELOW
       End { Else };
       TRYCARDINALPTS := RES
  END (* TRYCARDINALPTS *);

  FUNCTION FINDFIRSTFREEPOSITIONCLOSETO(inCOORD: COORDINATE ): BOXPTR;
  (* FIND THE CLOSEST POSITION TO COORD WHICH ISN'T OCCUPIED *)
  CONST NOOFATIMP = 5;
```

```
VAR XCOUNT, YCOUNT: INTEGER;
    SXINC,SYINC, XINC, YINC: INTEGER;
    BOX: BOXPTR;
    FREEPOSFOUND, SHOULDCONTINUE: BOOLEAN;
    I: INTEGER;

BEGIN

    (* NEED TO LEAVE A REASONABLE GAP BETWEEN ENTITIES AND ARE DEALING
       WITH CENTRE OF ENTITIES, SO NEED TO CONSIDER AT LEAST SIZE OF ONE
       BOX *)

    SXINC := XGAP+WIDTHOFENTITYBOX;
    SYINC := YGAP+HEIGHTOFENTITYBOX;
    XINC := XGAP;
    YINC := YGAP;
    FREEPOSFOUND := FALSE;
    (* IF THERE IS AN ENTITY RELATED TO THIS ONE, TRY CARDINAL PTS.  *)
    If (NOT TRYCARDINALPTS( INCOORD , SXINC , SYINC , BOX)) Then
    BEGIN
       SETBOXAT(BOX,(INCOORD.X-SXINC), (INCOORD.Y+SYINC) );
       WITH BOX↑ DO
       REPEAT
            YCOUNT := 1;
            REPEAT
                 XCOUNT := 1;
                 REPEAT
                      (* NO POINT TESTING IF IN SAME POSITION *)
                      IF same_coord(CENTRE↑ , INCOORD) OR
                         (NOT free_position( BOX ) ) THEN
                      BEGIN
                           setboxat(box,(centre↑.x+xinc),centre↑.y);
                           XCOUNT := XCOUNT + 1
                      END
                      ELSE
                           FREEPOSFOUND := TRUE;
                 UNTIL FREEPOSFOUND OR (XCOUNT = NOOFATTMP);
                 IF NOT FREEPOSFOUND THEN
                 BEGIN
                      (* RESET X TO ORIGINAL VALUE & CHANGE Y *)
                      setboxat(box,(centre↑.x-((noofattmp-1) * xinc)),
                                   (centre↑.y - yinc) );
                      YCOUNT := YCOUNT + 1
                 END (* THEN *)
            UNTIL FREEPOSFOUND OR (YCOUNT = NOOFATTMP);
            IF NOT FREEPOSFOUND THEN
            BEGIN
                 (* HAVE FAILED TO FIND A FREE SPACE,
                    SO TRY FURTHER OUT *)
                 (* FIRST RESET Y TO ORIGINAL VALUE *)
                setboxat(box,centre↑.x,(centre↑.y+((noofattmp-1)*yinc)));
                 XINC := XGAP+XINC;
                 YINC := YINC+YGAP
            END (* THEN *)
       UNTIL FREEPOSFOUND;
```

```
     END (* WITH *);
     FINDFIRSTFREEPOSITIONCLOSETO := BOX
END (* FINDFIRSTFREEPOSITIONCLOSETO *);

FUNCTION FINDPOSITIONFORENTITY( ENTPT: ENTITYPTR): BOXPTR;
(* FIND A FREE POSITION FOR SPECIFIED ENTITY *)
VAR POSOFRELATEDENTITY: COORDPTR;
    BOX: BOXPTR;
BEGIN
     (* FIRST SEE IF ANY OF THE ENTITES THIS ENTITY IS RELATED TO
        ARE PLOTTED *)
     POSOFRELATEDENTITY := pos_of_entity_related_TO( ENTPT );
     (* IF ONE IS FOUND, THEN PLACE THE ENTITY CLOSE TO IT *)
     IF posofrelatedentity <> NULLCOORD THEN
          FINDPOSITIONFORENTITY :=
               FINDFIRSTFREEPOSITIONCLOSETO( POSOFRELATEDENTITY↑ )
     ELSE IF  HDENTPLOT = NIL THEN
     BEGIN
          (* FIRST ENTITY, SPECIAL CASE PLACED AT ORIGIN *)
          FINDPOSITIONFORENTITY := setupbox( strtpt↑ );
     END
     ELSE
          (* PLACE AS CLOSE TO ORIGIN AS CAN BE FOUND *)
          FINDPOSITIONFORENTITY := FINDFIRSTFREEPOSITIONCLOSETO( orign↑ )
END (* FINDPOSITIONFORENTITY *);

PROCEDURE CONNECTUPRELATIONSHIPS( ERLST: ENTITYRELPTR;
                                  PRIMEENTITYBOX: BOXPTR ); FORWARD;

FUNCTION PLCEENTITY( EPPT: ENTPLOTPTR ): BOXPTR;
(* PLACE SPECIFIED ENTITY ON PLOT & CONNECT RELS *)
VAR HDRELLST: RELPLOTPTR;
BEGIN
     (* CHECK SOMETHING THERE *)
     IF EPPT<  >NIL THEN
     BEGIN
          WRITELN;
          WRITELN('NOW PLACING ENTITY - ',EPPT↑.ENTITY↑.ENTITYNAME);
          WRITELN;
          EPPT↑.BOXPLOT := FINDPOSITIONFORENTITY( EPPT↑.ENTITY );
          (* NOW TRANSFER ENTITY TO PLOTTED ENTITY LIST *)
          TRANSFRENTTOPLOTTEDLIST( EPPT );
          CONNECTUPRELATIONSHIPS( EPPT↑.ENTITY↑.RELATONSPTR ,
                                  EPPT↑.BOXPLOT );
          PLCEENTITY := EPPT↑.BOXPLOT
     END (* IF *)
     ELSE PLCEENTITY := NIL
END (* PLCEENTITY *);

FUNCTION PLOTINVOLUTEDREL( BOX: BOXPTR ): COORDLISTPTR;
(* TO PLOT AN INVOLUTED RELATION *)
BEGIN
     PLOTINVOLUTEDREL := PLOTRELATIONSHIP( BOX , BOX )
END (* PLOTINVOLUTEDREL *);
```

```
PROCEDURE CONNECTRELATIONSHIP( ERPT: ENTITYRELPTR;
                               PRIMEENTITYBOX: BOXPTR );
(* CONNECT PRIME ENTITY TO ENTITY ON OTHER END OF SPECIFIED REL
   IF THIS ENTITY ISN'T ON THE PLOT ALREADY, THEN PLACE IT *)
VAR RPPT: RELPLOTPTR;
    OTHEREPPT: ENTPLOTPTR;
    OTHERENT: ENTITYPTR;
    SRTDENT: ENTPLOTPTR;
    OTHERBOX: BOXPTR;
BEGIN
     (* FIND OTHER ENTITY/REL LINK OF OTHER ENTITY IN REL *)
     OTHERENT := FINDOTHERENT( ERPT );
     (* FIND COORD OF THIS ENTITY *)
     OTHEREPPT := FINDENTPLOTENTRY( OTHERENT );
     IF OTHEREPPT = NIL THEN
     BEGIN
          (* COORD DOESN'T EXIST *)
          SRTDENT := FINDSORTEDENTRY( OTHERENT );
          OTHERBOX := NIL;
          IF SRTDENT = NIL THEN
               WRITELN(OUT,'ERROR ENTITY - ',
                         ' IS NOT IN UNPLOTTED OR PLOTTED ENTITY LIST')
          ELSE If place_by_entity Then
               (* OTHER ENTITY NOT ON PLOT YET, SO PLACE IT *)
               OTHERBOX := PLCEENTITY( SRTDENT );
     END
     ELSE
          (* ENTITY PLOTTED, SO GET INFO ABOUT IT *)
          OTHERBOX := OTHEREPPT↑.BOXPLOT;
     IF OTHERBOX <>NIL THEN
     BEGIN
          (* OTHER ENTITY EXISTS *)
          (* FIRST ENSURE RELATIONSHIP HASN'T BEEN PLOTTED ALREADY,
             THIS CAN OCCUR DUE TO PLACING OF AN ENTITY IN PROCESS
             OF CONNECTING RELATIONSHIP, THIS WILH HAVE ITS
             RELATIONSHIPS CONNECTED, WHICH WILL INCLUDE THE ONE
             THAT CAUSED IT TO BE PLACED *)
          RPPT := FINDRELPLOTENTRY( ERPT↑.RELPT );
          IF RPPT = NIL THEN
          BEGIN
               (* NOT PLOTTED, SO FIND ENTRY IN UNPLOTTED LIST *)
               RPPT := FINDRELUNPLOTENTRY( ERPT↑.RELPT );
               IF RPPT = NIL THEN
                    WRITELN(OUT,'ERROR - NON-EXISTENT RELATION')
               ELSE BEGIN
                    (* HASN'T BEEN PLOTED AS YET, SO DO SO *)
                    (* CHECK IF IT AN INVOLUTED RELATIONSHIP *)
                    WRITELN('CONNECTING RELATIONSHIP - ');
                    WRITERELNAME( ERPT );
                    WRITELN;
                    IF otherbox = primeentitybox THEN
                       RPPT↑.CRDLIST := PLOTINVOLUTEDREL( PRIMEENTITYBOX )
                    ELSE IF ERPT↑.RELATONTYP = 'A' THEN
                       RPPT↑.CRDLIST := PLOTRELATIONSHIP( PRIMEENTITYBOX,
                                                          OTHERBOX )
```

```
                    ELSE
                       RPPT↑.CRDLIST := PLOTRELATIONSHIP( OTHERBOX ,
                                                          PRIMEENTITYBOX )
                END(* ELSE *);
                IF RPPT↑.CRDLIST <> NIL THEN
                     TRANSFRRELTOPLOTTEDLIST( RPPT );
            END (* ELSE *)
        END (* IF *)
    END (* CONNECTRELATIONSHIP *);

    PROCEDURE CONNECTUPRELATIONSHIPS(* ERLST: ENTITYRELPTR;
                                       PRIMEENPITYBOX: BOXPTR *);
    (* TO CONNECT UP ALL THE RELATIONSHIPS FOR THE ENTITY *)
    VAR ERLSTPT: ENTITYRELPTR;
    BEGIN
        (* GO DOWN LIST OF RALATKNSHIPS & CONNECT ALL THE INDIVIDUAL
           RELATIONSHIPS *)
        ERLSTPT := ERLST;
        WHILE ERLSTPT <> NIL DO
        BEGIN
            CONNECTRELATIONSHIP( ERLSTPT ,
                                 PRIMEENTITYBOX );
            (* COJJACP NAXT RALATIONSHIP *)
            ERLSTPT := ERLSTPT↑.NEXTRELFORENTITY
        END (* SHILE *)
    END (* CONNECTUPRELATIONSHIPS *);


BEGIN
    (* TAKE EACH ENTITY IN TURN FROM THE SORTED LIST AND PLACE IT IN THE
       PLOT *)
    MAINEPPT := HDSORTEDENT;
    WHILE MAINEPPT <> NIL DO
    BEGIN
        TBOX := PLCEENTITY( MAINEPPT );
        (* ENTITY WILL HAVE BEEN TRANSFERED, SO NEED HEAD OF LIST *)
        MAINEPPT := HDSORTEDENT
    END (* WHILE *)
END (* SEGMENT PLACEENTITIES *);

    Procedure write_out_plot_positions;
    { To write out the positions just plotted}

        Procedure write_entity_information;
        Var eppt: entplotptr;
        Begin
            eppt := hdentplot;
            While eppt <> Nil Do
            With eppt↑ Do
            Begin
                Writeln(pltfil, entity↑.entityname,
                                entity↑.ent_name_lth,
                                boxplot↑.xcoords[lessxbnd],
                                boxplot↑.ycoords[gtybnd],
                                boxplot↑.xcoords[gtxbnd],
```

```
                                boxplot↑.ycoords[lessybnd] );
            eppt := nextentry
        End {While With};

        Writeln(pltfil, end_of_entity_marker, 0, 0, 0, 0, 0)
    End {write_entity_information};

    Procedure write_rel_information;
    Var rppt: relplotptr;
        clp: coordlistptr;
        no_rels: Integer;

        Procedure write_edge( edge: rectedge );
        { To write an acceptable edge code}
        Begin
            Case edge Of
                top: Write(pltfil,'t');
                left: Write(pltfil,'l');
                bottom: Write(pltfil,'b');
                right: Write(pltfil,'r');
                niledge: Write(pltfil,'n')
            End {Case}
        End {write_edge};

    Begin
        rppt := hdrelplot;
        While rppt <>Nil Do
        With rppt↑ Do
        Begin
            Write(pltfil, relaton↑.activerelname,
                          relaton↑.activeptr↑.optality,
                          relaton↑.activeptr↑.degree,
                          relaton↑.passiverelname,
                          relaton↑.passiveptr↑.optality,
                          relaton↑.passiveptr↑.degree );
            no_rels := 0;
            clp := crdlist;
            While clp< >Nil Do
            Begin
                no_rels := no_rels + 1;
                clp := clp↑.nextcoord
            End;
            Writeln(pltfil, no_rels);

            clp := crdlist;
            While clp <>Nil Do
            With clp↑ Do
            Begin
                Write(pltfil, startcoord↑.x,startcoord↑.y,
                              endcoord↑.x,endcoord↑.y,' ',
                              linegrad,' ',
                              lineconst);
                write_edge( strtedge );
                write_edge( endedge );
                Writeln(pltfil);
```

```
                    clp := nextcoord
                End {While With};
                rppt := nextentry
            End {While With}
        End {write_rel_information};
    Begin
        Rewrite( pltfil );
        write_entity_information;
        write_rel_information
    End {write_out_plot_positions};


    Procedure get_options;
    Var reply: Char;
    Begin
        sorted := False;
        ascending := False;
        average_rel := False;
        place_by_entity := False;
        Write('Do you wish Relationship Priority? ');
        Write(' (y/n) - ');
        Readln(reply);
        If (reply = 'y')  Or (reply = 'Y') Then
        Begin
            {can't have placing by entity}
            sorted := True;
            Write('Do you wish them placed in ascending order? (y/n) - ');
            Readln( reply );
            If (reply = 'y') Or (reply = 'Y') Then ascending := True;
            Write('You have chosen placing by ');
            If ascending Then Write('ascending ') Else Write('descending ');
            Writeln('no. of relationships.')
        End;
        Write('Do you wish Entity Priority ');
        If sorted Then Write('as well ? ') Else Write('? ');
        Readln( reply );
        If (reply = 'y') Or (reply = 'Y') Then place_by_entity := True;
        Write('You have chosen placing by giving ');
        If place_by_entity Then Writeln('entities ')
                           Else Writeln('relationships ');
        Write('most importance');
        If sorted Then Writeln(' as well as sorting.') Else Writeln('.');
        Write('Do you wish Average Entity Proximity ? ');
        Readln( reply );
        If (reply = 'y') Or (reply = 'Y') Then average_rel := True;
    End {get_options};

BEGIN
    WRITELN('              DIAGRAMMER              ');
    WRITELN;
    WRITELN;
    get_options;
    INITIALISE;
    WRITELN;
    WRITELN('THE MODEL IS NOW BEING CONSTRUCTED');
```

```
    PLACEENTITIES;
    WRITELN;
    WRITELN('The model positions are now being written out');
    write_out_plot_positions
END (* PLOT *).
```

```
Program draw(pltfil, Input, Output);
{ A program to draw a picture which has been plotted by PLOT}


Const infinite_grad = Maxint;
      end_of_entity_marker = '$$$$$                                    ';
      widthofentitybox = 8;
      heightofentitybox = 4;
      maxnamlth = 32;   { All these values from PLOT program }

Type name = String( maxnamlth );
     rectedge = (niledge, top, right, bottom, left);
     component = Integer;
     opttyp = 'M'..'O';
     degree = %M'.'O';
     coordinate = Record
          x: component;
          y: component
     End {coordinate};
     { All above values from PLOT }

     entrec = Record
          ent_name: name;
          ent_name_lth: Integer;
          x_lht,
          y_lht,
          x_rhb,
          y_rhb: component
      End {entrec};

      relrec = Record
          act_rel_name: name;
          act_rel_opt: opttyp;
          act_rel_deg: degree;
          pas_rel_name: name;
          pas_rel_opt: opttyp;
          pas_rel_deg: degree;
          no_of_rels: Integer
      End {relrec};

      relval = Record
           start_coord:coordinate;
           end_coord: coordinate;
           line_grad: Real;
           line_const: Real;
           strt_edge: rectedge;
           end_edge: rectedge
      End {relval};

      entpt = ^ents;
      relpt = ^rels;
      pathpt = ^paths;

      ents = Record
           ent: entrec;
```

```
          next_ent: entpt
     End {ents};

     rels = Record
          rel: relrec;
          paths: pathpt;
          next_rel: relpt
     End {rels};

     paths = Record
          path: relval;
          next_path: pathpt
     End {paths};


Var  hdents, tlents: entpt;
     hdrels, tlrels: relpt;
     ent: entrec;
     rel: relrec;
     pltfil: Text;
     optality_on, scale_changed, box_on: Boolean;

Procedure read_stuff;
Var path_val: relval;

     Procedure read_name( Var nam: name );
     { To read a name from pltfil }
     Var i: Integer;
     Begin
          {Assume name is complete at maxnamlth chars long}
          For i := 1 To maxnamlth Do
               If Not Eof(pltfil) Then Read(pltfil , nam[i])
     End {read_name};

     Procedure read_ent;
     Begin
          With ent Do
          Begin
               {First read name}
               read_name( ent_name );
               {Next read rest of stuff}
               If Not Eof(pltfil) Then Readln( pltfil , ent_name_lth,
                                               x_lht,
                                               y_lht,
                                               x_rhb,
                                               y_rhb )
          End {With}
     End {read_ent};

     Procedure read_rel;
     Begin
          With rel Do
          Begin
               read_name( act_rel_name );
               If Not Eof(pltfil) Then
```

```
                Read(pltfil, act_rel_opt , act_rel_deg );
            If Not Eof(pltfil) Then read_name( pas_rel_name );
            If Not Eof(pltfil) Then
                Read(pltfil , pas_rel_opt , pas_rel_deg );
            If Not Eof(pltfil) Then Readln( pltfil , no_of_rels )
        End {With}
    End {read_rel};

    Procedure read_edge( Var edge: rectedge );
    { To read an edge code & translate into an edge }
    Var e_val: char;
    Begin
        Read( pltfil , e_val );
        Case e_val Of
            'n': edge := niledge;
            'r': edge := right;
            'l': edge := left;
            't': edge := top;
            'b': edge := bottom
        End {Case}
    End {read_edge};

    Procedure read_path;
    { To read a path  value from file }
    Begin
        With path_val Do
        Begin
            Read( pltfil, start_coord.x , start_coord.y,
                          end_coord.x, end_coord.y,
                          line_grad, line_const );
            If Not Eof(pltfil) Then read_edge( strt_edge );
            If Not Eof(pltfil) Then read_edge( end_edge );
            If Not Eof(pltfil) Then Readln(pltfil)
        End {With}
    End {read_path};

    Procedure read_entity_information;
    {To read all entity information from file}
    Begin
        hdents := Nil;
        read_ent; { read var. 'ent' }
        While Not Eof(pltfil) And (ent.ent_name<>end_of_entity_marker) Do
        Begin
            { Set up entry }
            If hdents = Nil Then
            Begin
                New( hdents );
                tlents := hdents
            End
            Else Begin
                New( tlents^.next_ent );
                tlents := tlents^.next_ent
            End;

            tlents^.ent := ent;
```

```
            read_ent
        End {While };
        tlents^.next_ent := Nil
    End {read_entity_information};

    Function read_paths( no_of_paths: Integer ): pathpt;
    Var hdpaths, tlpaths: pathpt;
        i: Integer;
    Begin
        i := 1;
        hdpaths := Nil;
        If no_of_paths<>0 Then
        Begin
           read_path; { read  var. 'path_val' }
           Repeat
             If hdpaths = Nil Then
             Begin
                  New( hdpaths );
                  tlpaths := hdpaths
             End
             Else Begin
                  New( tlpaths^.next_path );
                  tlpaths := tlpaths^.next_path
             End;

             tlpaths^.path := path_val;
             i := i+1;
             If i <= no_of_paths Then read_path
           Until (i>no_of_paths);
           tlpaths^.next_path := Nil
        End {Then};

        read_paths := hdpaths
    End {read_paths};

    Procedure read_rel_information;
    { To read relation information from file }
    Begin
        hdrels := Nil;
        read_rel; { read var. 'rel' }
        While Not Eof(pltfil) Do
        Begin
             { Set up entry }
             If hdrels = Nil Then
             Begin
                  New( hdrels );
                  tlrels := hdrels
             End
             Else Begin
                  New( tlrels^.next_rel );
                  tlrels := tlrels^.next_rel
             End;

             tlrels^.rel := rel;
             tlrels^.paths := read_paths( rel.no_of_rels );
```

```
                read_rel
            End {While };
            tlrels^.next_rel := Nil
        End {read_rel_information};

Begin
     Reset( pltfil );
     read_entity_information;
     read_rel_information
End;


PROCEDURE draw_plot;
CONST XCENTRE = 400;
      YCENTRE = 200;
      screen_top_edge = 0;
      screen_bottom_edge = 399;
      left_edge_after_menu = 120;
      screen_right_edge = 799;
      escape = Chr(27);
      cr = Chr(160)*Chr(161);
      grafix = escape*'5';
      bigchlth = 10;
      smallchlth = 6;
      smallchht = 8;
VAR MAXX, MAXY, MINX , MINY: COMPONENT;
    CH,OCH, curr_x_dir, curr_y_dir: CHAR;
    XFAC, YFAC, OXFAC, OYFAC, SPEED,
    XSCALE, OXSCALE, YSCALE, OYSCALE: INTEGER;
    screen_left_edge,i,no_of_times: Integer;
    leftedge, rightedge, topedge, bottomedge: Integer;
    MENUON, initialisation, small, opt_set: BOOLEAN;
    radius,xcurve,ycurve,length,height,min_x,min_y: Integer;
     Procedure write_entity_shape(x1,y1,x2,y2: Integer);
     Begin
          Writeln(grafix,'Q',x1+xcurve,y2);
          Writeln(grafix,'U',x2-xcurve,y2);
          Writeln(grafix,'Q',x2-xcurve-2,y2+ycurve+2);
          Writeln(grafix,'h',radius,x2,y2+ycurve,x2-xcurve,y2);
          Writeln(grafix,'Q',x2,y2+ycurve);
          Writeln(grafix,'U',x2,y1-ycurve);
          Writeln(grafix,'Q',x2-xcurve-2,y1-ycurve-2);
          Writeln(grafix,'h',radius,x2-xcurve,y1,x2,y1-ycurve);
          Writeln(grafix,'Q',x2-xcurve,y1);
          Writeln(grafix,'U',x1+xcurve,y1);
          Writeln(grafix,'Q',x1+xcurve+2,y1-ycurve-2);
          Writeln(grafix,'h',radius,x1,y1-ycurve,x1+xcurve,y1);
          Writeln(grafix,'Q',x1,y1-ycurve);
          Writeln(grafix,'U',x1,y2+ycurve);
          Writeln(grafix,'Q',x1+xcurve+2,y2+ycurve+2);
          Writeln(grafix,'h',radius,x1+xcurve,y2,x1,y2+ycurve);
          box_on := True
     End { Procedure write_entity_shape};

     Procedure draw_name( entname: name;
```

```
                    ent_name_lth: Integer;
                    x_lht, y_lht: component);
Var nam: name;
    i,j, namlth, no_of_posns, ch_ht: Integer;
Const xstrt = 2;

Begin
     {First move cursor to right posn}
     Writeln(grafix,'Q',x_lht+xstrt,y_lht+yscale);
     If (xscale > 15) And (ent_name_lth*bigchlth <= (length-xstrt)) Then
     Begin
          If small Then
          Begin
               Writeln(grafix,'iNORMAL');
               small := False
          End;
          Writeln(grafix,'p',entname:ent_name_lth)
     End
     Else Begin
          If Not small Then
          Begin
               Writeln(grafix,'ismall');
               small := True
          End;
          If ent_name_lth*smallchlth <= (length-xstrt) Then
               Writeln(grafix,'p',entname:ent_name_lth)
          Else Begin
               Writeln(grafix,'R',0,-3-(yscale div 2));
               i := 1;
               no_of_posns := (length div smallchlth) - 1;
               While (i<=no_of_posns) And (entname[i] <> ' ') Do
               Begin
                    nam[i] := entname[i];
                    i := i + 1;
               End;
               Writeln(grafix,'p',nam:(i-1));
               Writeln(grafix,'R',-((i-1)*smallchlth),smallchht+4);
               While (i<=ent_name_lth) And (entname[i] <> ' ') Do
                    i := i + 1;
               While (i<=ent_name_lth) And (entname[i] = ' ') Do
                    i := i + 1;
               If i <= ent_name_lth  Then
               Begin
                    If ent_name_lth <= (i+no_of_posns) Then
                         namlth := ent_name_lth-i+1
                    Else namlth := no_of_posns;
                    For j := 1 To namlth Do
                         nam[j] := entname[i-1+j];

                    Writeln(grafix,'p',nam:namlth);
               End
          End
     End {Else}
End {draw_name};
```

```
Procedure drawbx( ent_val: entrec );
Begin
     With ent_val Do
     Begin
          { Move cursor to lh. top corner }
          x_lht := (x_lht+xfac);
          y_lht := (y_lht+yfac);
          x_rhb := (x_rhb+xfac);
          y_rhb := (y_rhb+yfac);
          If ((x_lht >= leftedge) And (x_lht <= rightedge) And
              (y_lht <= bottomedge) And (y_lht >= topedge)) Or
             ((x_rhb >= leftedge) And (x_rhb <= rightedge) And
              (y_rhb <= bottomedge) And (y_rhb >= topedge))   Then
          Begin
               Writeln(grafix,'Q',x_lht,y_lht);
               {Move window into position}
               Writeln(grafix,'A1');
               Writeln(grafix,'V0');
               draw_name( ent_name , ent_name_lth , x_lht , y_lht )
          End {If};
     End {With};
End {Drawbx};

PROCEDURE DRAWMANY( st,en: coordinate;
                    M , C: REAL;
                    EDGE: RECTEDGE);
(* MIDDLE LINE OF CROWS FOOT ALREADY THERE, HAVE TO DRAW SIDES *)
VAR DELTAX , DELTAY: COMPONENT;
    x1, y1, x2, y2: component;
BEGIN
  x1 := (st.x+xfac);
  y1 := (st.y+yfac);
  x2 := (en.x+xfac);
  y2 := (en.y+yfac);
  IF ((X1>=LEFTEDGE) AND (X1<=RIGHTEDGE)) AND
     ((Y1<=BOTTOMEDGE) AND (Y1>=TOPEDGE)) THEN
  BEGIN
     IF M = infinite_grad THEN DELTAX := 0
     ELSE DELTAX := ROUND(XSCALE * (1 / SQRT((M*M) + 1)));
     IF X1 > X2 THEN DELTAX := - DELTAX;
     IF M = infinite_grad THEN
     BEGIN
          IF Y1 > Y2 THEN DELTAY := +YSCALE{for a normal screen other}
                     ELSE DELTAY := -YSCALE{way round}
     END
     ELSE DELTAY := rOUND( M * DELTAX );
     IF (EDGE = TOP) OR (EDGE = BOTTOM) THEN
     BEGIN
          Writeln(grafix,'Q', X1-(XSCALE DIV 2) , Y1 );
          Writeln(grafix,'U', X1+DELTAX , Y1-DELTAY );
          Writeln(grafix,'U', X1 + (XSCALE DIV 2) , Y1 )
     END
     ELSE BEGIN
          Writeln(grafix, 'Q',X1 , Y1-(yscALE DIV 2) );
          Writeln(grafix,'U', X1+DELTAX , Y1-DELTAY );
```

```
          Writeln(grafix,'U', X1 , Y1+(YSCALE DIV 2) )
      END (* ELSE *)
  END
END (* DRAWMANY *);

PROCEDURE DRAWREL( X1,Y1, X2,Y2: COMPONENT; opt: opttyp);
BEGIN
  IF (((X1>=LEFTEDGE) AND (X1<=RIGHTEDGE)) AND
      ((Y1<=BOTTOMEDGE) AND (Y1>=TOPEDGE))    ) OR
     (((X2>=LEFTEDGE) AND (X2<=RIGHTEDGE))    AND
      ((Y2<=BOTTOMEDGE) AND (Y2>=TOPEDGE))    ) THEN
  Begin
        If optality_on Then
        Begin
             If (opt = 'O') And (Not opt_set) Then
             Begin
                  Writeln( grafix , 'Z2' );
                  opt_set := True
             End Else
             If (opt = 'M') And opt_set Then
             Begin
                  Writeln( grafix , 'Z1' );
                  opt_set := False
             End
        End;
        Writeln(grafix,'Q',x1,y1);
        Writeln(grafix,'U',X2 , Y2 );
  End;
END (* DRAWREL *);

PROCEDURE DRAWR( ST ,EN: COORDINATE; opt: opttyp);
BEGIN
     DRAWREL((ST.X+xfac),(ST.Y+yfac),
             (EN.X+xfac),(EN.Y+yfac), opt)
END (* DRWAR *);



PROCEDURE draw_entities;
(* TO draw ENTITY PLOT POSITIONS *)
VAR ept: entpt;
    I: INTEGER;
BEGIN
     (* NEXT GO DOWN LIST OF ENTITIES WRITING INFO TO FILE *)
     ept := hdents;
     WHILE ept    NIL DO
     BEGIN
          DRAWBX( ept^.ent );
          (* GET NEXT ENTITY *)
          ept := ept^.next_ent
     END (* WHILE *)
END (* draw_entities*);

Function draw_relationship( ppt: pathpt;
                            start_no, end_no: Integer;
```

```
                                opt: opttyp   ): pathpt;
{ To draw all parts of a relationship }
Var tppt: pathpt;
    i: Integer;
Begin
     tppt := ppt;
     For i:= start_no To  end_no-1 Do
     With tppt↑.path Do
     Begin
          drawr( start_coord ,end_coord , opt);
          tppt := tppt↑.next_path
     End {With For};
     { Are at last rel in  list }
     With tppt↑.path Do drawr( start_coord , end_coord , opt);
     draw_relationship := tppt
End {draw_relationship};

Procedure draw_half_opt_single_rel( rl: relpt );
{ To draw a single line which is half optional }
Var min_x, min_y, mid_x , mid_y, x_diff, y_diff: Integer;
Begin
     With rl↑.paths↑.path Do
     Begin
          { First find mid point of line ( must allow for integers ) }
          If end_coord.x < start_coord.x Then min_x := end_coord.x
                                         Else min_x := start_coord.x;
          If end_coord.y < start_coord.y Then min_y := end_coord.y
                                         Else min_y := start_coord.y;
          x_diff := Abs(start_coord.x - end_coord.x);
          y_diff := Abs(start_coord.y - end_coord.y);
          If line_grad = 0 Then mid_y := min_y
          Else If Odd( y_diff ) Then mid_y := min_y+((y_diff-1) Div 2)
                                Else mid_y := min_y +(y_diff Div 2);
          If line_grad = maxint Then mid_x := min_x
          Else If line_grad = 0 Then mid_x := min_x + (x_diff Div 2)
          Else
              {y is negative due to vagaries of Sirius screen}
              mid_x :=
                       min_x + (x_diff div 2);
          drawrel((start_coord.x+xfac) ,(start_coord.y + yfac) ,
                  (mid_x + xfac) , (mid_y + yfac),
                 rl↑.rel.act_rel_opt );
          drawrel( (mid_x + xfac) ,( mid_y + yfac) ,
                  (end_coord.x + xfac) , (end_coord.y + yfac),
                  rl↑.rel.pas_rel_opt );
     End {With}
End {draw_half_opt_single_line};

Procedure draw_a_relationship( rl: relpt );
{ To arrange the drawing of a relationship }
Var end_rel: pathpt;
Begin
     opt_set := False;
     With rl↑.rel Do
     Begin
```

```
            IF act_rel_deg = 'M' THEN
                 With rl↑.paths↑.path Do
                 DRAWMANY( START_COORD , END_COORD,
                           LINE_GRAD, LINE_CONST,
                           STRT_EDGE );
            If (act_rel_opt = pas_rel_opt) Or Not optality_on Then
                 { Must be completely optional or mandatory }
                 end_rel := draw_relationship(rl↑.paths,1,no_of_rels,
                                              act_rel_opt)
            Else If no_of_rels = 1 Then
            Begin
                 draw_half_opt_single_rel( rl );
                 end_rel := rl↑. paths
            End
            Else Begin
                 { Draw relationship half way & set end_rel to last path
                   drawn }
                 end_rel := draw_relationship( rl↑.paths , 1,
                                              (no_of_rels Div 2),
                                              act_rel_opt );
                 end_rel := draw_relationship( end_rel↑.next_path ,
                                              (no_of_rels Div 2)+1,
                                              no_of_rels, pas_rel_opt )
             End { Else };
             If opt_set Then
             Begin
                  { reset dotted lines to full }
                  Writeln( grafix , 'Z1' );
                  opt_set := False
             End;
             { Draw end many if necessary }
             IF pas_rel_deg = 'M' THEN
                  With end_rel↑.path Do
                      DRAWMANY( END_COORD , START_COORD,
                                LINE_GRAD ,
                                LINE_CONST, END_EDGE)
      End {With}
End {draw_a_relationship};

PROCEDURE draw_rels;
(* NOW DO THE SAME FOR RELATIONSHIPS *)
VAR rpt: relpt;
    ppt: pathpt;
BEGIN
     rpt := hdrels;
     WHILE rpt <> NIL DO
     BEGIN
          draw_a_relationship( rpt );
          rpt := rpt↑.next_rel
     END (* WHILE WITH RPPT *)
END (* draw_rels *);


PROCEDURE WRITEMENU;
BEGIN
```

```
        Writeln(grafix,'Q',0,0);
        Writeln(grafix,'iNORMAL');
        { First set complete band to reverse video}
        Writeln(grafix,'I',left_edge_after_menu,screen_bottom_edge);
        Writeln(grafix,'X15');
        Writeln(grafix,'V0');
        Writeln(grafix,'v');{Reverse Video}
        Writeln(grafix,'p',cr,'F1 Quit ',cr,
        'F2 Prnt Scrn',cr,
        'F3 Reset',cr,
        'F4 Opt On',cr,
        'F5 Opt Off',cr,
        'F6 Zoom In',cr,
        'F7 Zoom Out',cr,
        'Precede By ',cr,
        'no. of times',cr,
        'KEYPAD FNS',cr,
        '  Scroll Up',cr,
        '       Down',cr,
        '       Left',cr,
        '      Right',cr,
        '  SHIFT  ',cr,
        '  MOVES 1/2',cr,
        '  SCREEN',cr,
        'ESC Menu',cr,
        '    On/Off',cr,
        'CONT - chge',cr,
        'VIEW MODE'
        );
        Writeln(grafix,'w');
        Writeln(grafix,'iSMALL');
    END (* WRITEMENU *);

    Procedure draw_mod;
    Begin
      {Clear drawing screen}
      Writeln(grafix,'Q',leftedge,topedge);
      Writeln(grafix,'I',(rightedge-leftedge),(bottomedge-topedge));
      Writeln(grafix,'X0');
      Writeln(grafix,'V0');
      Writeln(grafix,'X7');
      {Reset work screen window}
      Writeln(grafix,'A1');
      Writeln(grafix,'Q',0,0); {Reset shape window}
      Writeln(grafix,'I',length+1,height+1);
      If (Not box_on) Or (scale_changed) Then
      Begin
           {produce new box shape}
           Writeln(grafix,'2');{clear screen 1}
           write_entity_shape(0,height,length,0);
      End;
      Writeln(grafix,'A0');
      draw_entities;
      draw_rels
    End {draw_mod};
```

```
Procedure move_drawing( xdir, ydir: Char;
                        dist : Integer);
{ To move a drawing dist in given dirs }
Var move_wind_lhx, move_wind_lhy,
    new_lhx, new_lhy,
    xdist, ydist: Integer;
Begin
     curr_x_dir := xdir;
     curr_y_dir := ydir;
     xdist := 0; ydist := 0;
     move_wind_lhx := screen_left_edge; move_wind_lhy := screen_top_edge;
     new_lhx := screen_left_edge; new_lhy := screen_top_edge;
     {set values for screen move to achieve desired direction}
     If (xdir = 'L') And (xfac <= (Maxint - dist)) Then
     Begin
          xfac := xfac - dist;
          xdist := dist;
          move_wind_lhx := screen_left_edge + dist
     End
     Else If (xdir = 'R') And (xfac >= (-Maxint + dist)) Then
     Begin
          xfac := xfac + dist;
          xdist := dist;
          new_lhx := screen_left_edge + dist
     End;
     If (ydir = 'D') And (yfac <= (maxint - dist)) Then
     Begin
          yfac := yfac + dist;
          ydist := dist;
          new_lhy := screen_top_edge + dist
     End
     Else If (ydir = 'U') And (yfac >= (-Maxint + dist)) Then
     Begin
          yfac := yfac - dist;
          ydist := dist;
          move_wind_lhy := screen_top_edge + dist
     End;
     {First move existing drawing}
     Writeln(grafix,'X3');
     {move to lh corner of part of model to be moved}
     Writeln(grafix,'Q',move_wind_lhx,move_wind_lhy);
     Writeln(grafix,'I',(rightedge-leftedge-xdist),
                        (bottomedge-ydist));
     {move to new posn of model}
     Writeln( grafix , 'Q' , new_lhx , new_lhy );
     Writeln( grafix , 'V0');
     {set screen edge vars to isolate strip to be done}
     If xdir = 'L' Then leftedge := rightedge - xdist
     Else If xdir = 'R' Then rightedge := leftedge + xdist;
     If ydir = 'U' Then topedge := bottomedge - ydist
     Else If ydir = 'D' Then bottomedge := topedge + ydist;
     { Draw in rest of model }
     draw_mod;
     { Reset screen edge vars. }
```

```
        rightedge := screen_right_edge;
        topedge := screen_top_edge;
        bottomedge := screen_bottom_edge;
        If menuon Then leftedge := left_edge_after_menu
                  Else leftedge := screen_left_edge;
    End {move_drawing};

    Procedure change_posns_scale(xdiff,ydiff: Integer);
    VAR ept: entpt;
        rpt: relpt;
        ppt: pathpt;
        Function change_value(v,scle,diff: Integer): Integer;
        Begin
            If initialisation Then change_value :=(v *scle)
            Else If scle = diff Then change_value := 0
            Else change_value :=Round((v  / (scle-diff))*scle);
        End;
    BEGIN
        (* NEXT GO DOWN LIST OF ENTITIES changing scale *)
        ept := hdents;
        WHILE ept<>Nil Do
        WITH EPT^.ent DO
        BEGIN
            x_lht := change_value( x_lht, xscale, xdiff);
            y_lht := change_value( y_lht, yscale , ydiff );
            x_rhb := change_value( x_rhb , xscale , xdiff );
            y_rhb := change_value( y_rhb , yscale , ydiff );
            (* GET NEXT ENTITY *)
            ept := ept^.next_ent
        END (* WHILE *);
        (* NOW DO THE SAME FOR RELATIONSHIPS *)
        rpt := hdrels;
        WHILE RPT <>NIL DO
        WITH RPT^ DO
        BEGIN
            (* Scale coordinate list *)
            ppt := paths;
            WHILE ppt <>NIL DO
            With ppt^.path Do
            BEGIN
                start_coord.x :=change_value(start_coord.x,xscale,xdiff);
                start_coord.y :=change_value(start_coord.y,yscale,ydiff);
                end_coord.x := change_value(end_coord.x,xscale,xdiff);
                end_coord.y := change_value(end_coord.y,yscale,ydiff);
                ppt := ppt^.next_path
            END(* While WITH *);
            (* GET NEXT RELATIONSHIP *)
            rpt := next_rel
        END (* WHILE WITH RPT *)
    END (* change_posns_scale *);

    Procedure change_scale(xdiff , ydiff: Integer);
    Begin
        If ((xscale + xdiff) = 0) Or ((yscale + ydiff) = 0) Then
            Writeln('Error - Invalid scaling, too small')
```

```
        Else Begin
             xscale := xscale+xdiff;
             yscale := yscale+ydiff;
             If Not initialisation Then change_posns_scale(xdiff,ydiff)
             Else Begin
                  yscale := -yscale;
                  change_posns_scale(xdiff,ydiff);
                  yscale := -yscale
             End;
             length := widthofentitybox * xscale;
             height := heightofentitybox * yscale;
             {Set Window to change box shape}
             scale_changed := true;
             draw_mod;
             scale_changed := false;
        End
   End;

   Procedure initialise;
   Begin
        initialisation := True;
        Writeln(grafix,'d'); {initialise screen}
        small := False;
        Writeln(escape,'m2',24,24);
        xscale := 0; yscale := 0;
        radius := 6;
        xcurve := 7;
        ycurve := 4;
        {Clear Screens}
        Writeln(grafix,'A0');
        Writeln(grafix,'2');
        Writeln(grafix,'B0');
        Writeln(grafix,'r');{Disable Cursor}
        WRITEMENU;
        menuon := True;
        optality_on := True;
        scale_changed := True;
        box_on := False;
        opt_set := False;
        screen_left_edge := left_edge_after_menu;
        leftedge := screen_left_edge;
        rightedge := screen_right_edge;
        topedge := screen_top_edge;
        change_scale(10,9); {draws model as well}
        bottomedge := screen_bottom_edge;
        XFAC := screen_left_edge+(length div 2); YFAC := ycentre;
        SPEED := 0;
        curr_x_dir := ' '; curr_y_dir := ' ';
        initialisation := False;
        draw_mod
   End;

   Function number( ch: Char ): Boolean;
   Begin
        number := (ch = '1') Or (ch = '2') Or (ch = '3') Or (ch = '4') Or
```

```
                (ch = '5') Or (ch = '6') Or (ch = '7') Or (ch = '8') Or
                (ch = '9') Or (ch = '0')
End {number};

Function convert( ch: char ): Integer;
Var res: Integer;
Begin
     Case ch Of
          '0': res := 0;
          '1': res := 1;
          '2': res := 2;
          '3': res := 3;
          '4': res := 4;
          '5': res := 5;
          '6': res := 6;
          '7': res := 7;
          '8': res := 8;
          '9': res := 9
     End { Case };
     convert := res
End {Convert};


Procedure get_command;
Var line: string(81);
    linepos: Integer;
Begin
     Readln( line );
     linepos := 1;
     While (linepos <= 80) And (line[ linepos ] = ' ') Do
          linepos := linepos + 1;
     If linepos > 80 Then ch := ' '
     Else Begin
          If (linepos > 80) Or Not number(line[linepos]) Then
               no_of_times := 1
          Else Begin
              no_of_times := 0;
              Repeat
                   no_of_times :=  (no_of_times*10) +
                                   convert( line [ linepos ]);
                   linepos := linepos  + 1
              Until (linepos > 80) Or (Not number(line[linepos]));
              While (linepos <= 80) And (line[ linepos ] = ' ') Do
                  linepos := linepos + 1;
          End;
          If linepos > 80 Then ch := ' ' Else ch := line [ linepos ];
     End { Else };
End {get_command};

Procedure interactive_move_mode;
{ To allow the movement of the model components interactively }
Var x_cur, y_cur: Integer;  { x & y absolute cursor position }
    move_ent: entpt; { pointer to entity to be moved }
    new_rel_pt: relpt;
    start_amend_path: pathpt;
```

```
no_of_paths: Integer;
latest_path_pt: pathpt;
first_new_path: pathpt;
rel_strt_edge: rectedge;
old_dir: Char; {vars to enable the draing of rels}

Procedure move_intialise;
{ Initialise things for interactive mode }
Begin
     { Choose & enable cross-hair cursor }
     Writeln(grafix,'xl');
     Writeln(grafix,'q');
     { position cursor in centre of screen }
     x_cur := xcentre; y_cur := ycentre;
     Writeln(grafix,'Q',xcentre,ycentre);
     move_ent := Nil; {No entity currently being moved}
     new_rel_pt := Nil;
     start_amend_path := Nil;
     no_of_paths := 0;
     latest_path_pt := Nil;
     first_new_path := Nil;
     old_dir := ' '; {no rel currently being drawn}
     Writeln(grafix,'A',0);
     Writeln(grafix,'B',0); {Work & display screens 0}
End {move_initialise};

Function on_path( relv: relval;
                  x_rel_pos, y_rel_pos: Integer)
                              : Boolean;
{ To see if cursor is on path }
Var res:Boolean;
    gt_x,gt_y,less_x,less_y: component;
Begin
     With relv Do
     Begin
          {first set up which comps are gtest & least}
          If start_coord.x > end_coord.x Then
          Begin
               gt_x := start_coord.x; less_x := end_coord.x
          End Else
          Begin
               gt_x := end_coord.x; less_x := start_coord.x
          End;
          If start_coord.y > end_coord.y Then
          Begin
               gt_y := start_coord.y; less_y := end_coord.y
          End Else
          Begin
               gt_y := end_coord.y; less_y := start_coord.y
          End;
          res := False;
          If (x_rel_pos >= less_x) And (x_rel_pos <= gt_x) And
             (y_rel_pos >= less_y) And (y_rel_pos <= gt_y) Then
          Begin
               {is within bounds of line, now must see if on line}
```

```
                    If line_grad = infinite_grad Then
                       res := (x_rel_pos = less_x)
                    Else If line_grad = 0 Then
                         {Horizontal line}
                         res := (y_rel_pos = less_y )
                    Else {Is diagonal line, use y=mx + c}
                         res :=
                          (Round((end_coord.y-y_rel_pos) / line_grad)=
                                (end_coord.x-x_rel_pos));
               End{If}
          End {With};
          on_path := res
     End{on_path};

     Function find_path( rpt: relpt;
                         x_rel_pos, y_rel_pos: Integer;
                         Var path_no: Integer): pathpt;
     {To find a path pointed at by cursor, returns path no}
     Var t_p_pt: pathpt;
         found: Boolean;
     Begin
          {look at paths to see if found}
          t_p_pt := rpt^.paths;
          path_no := 1; found := False;
          While (t_p_pt <> Nil) And (Not found) Do
          Begin
               If on_path(t_p_pt^.path,
                          x_rel_pos,y_rel_pos) Then
                    found := True
               Else Begin
                    {try next path}
                    t_p_pt := t_p_pt^.next_path;
                    path_no := path_no + 1
               End
          End;
          find_path := t_p_pt
     End {find_path};

     Function find_rel(Var pas:Boolean): relpt;
     {To find the rel pointed at by cursor }
     Var t_r_pt: relpt;
         t_p_pt: pathpt;
         x_rel_pos, y_rel_pos, x_pos,y_pos: Integer;
         path_no: Integer;
     Begin
          { Set screen free position of cursor }
          x_rel_pos := x_cur - xfac;
          y_rel_pos := y_cur - yfac;
          t_p_pt := Nil;
          t_r_pt := hdrels; {start at head of rel list}
          While (t_r_pt <> Nil) And (t_p_pt = Nil) Do
          Begin
               t_p_pt := find_path(t_r_pt, x_rel_pos, y_rel_pos,path_no);
               If t_p_pt = Nil Then
                    {isn't this rel, try next rel}
```

```
                t_r_pt := t_r_pt^.next_rel;
        End {While};
        {decide which end of rel is closest}
        pas := False; {is active end by default}
        If (t_r_pt <> Nil) Then
        If (t_r_pt^.rel.no_of_rels = 1) Then
        With t_r_pt^.paths^.path Do
          pas :=
             ((Abs(line_grad) > 1) And
        (Abs(y_rel_pos-end_coord.y)<Abs(y_rel_pos-start_coord.y))) Or
                ((Abs(line_grad)< 1) And
        (Abs(x_rel_pos-end_coord.x)<Abs(x_rel_pos-start_coord.x)))
        Else pas := (t_r_pt^.rel.no_of_rels-path_no)<(path_no);
        {have either found rel or cursor isn't properly
         positioned}
        If t_r_pt = Nil Then
             Write('Error - no relationship at this position');
        find_rel := t_r_pt
End {find_rel};


Function find_entity(Var edge: rectedge): entpt;
{ To find the entity pointed at by the cursor }
Var t_e_pt: entpt;
    x_rel_pos, y_rel_pos: Integer;
    found: Boolean;
Begin
     { Set screen free position of cursor }
     x_rel_pos := x_cur - xfac;
     y_rel_pos := y_cur - yfac;
     found := False;
     edge := niledge;
     t_e_pt := hdents; {start at head of entity list}
     While (t_e_pt <> Nil) And (Not found) Do
        {if isn't entity then inspect next entity}
        {First see if is on any of edges, if not then
          see if in confines of box, if not then isn't right}
        With t_e_pt^.ent Do
          If (y_rel_pos =y_lht) And
             (x_rel_pos >= x_lht) And (x_rel_pos <= x_rhb) Then
          Begin found:= True; edge:= top End
          Else If (y_rel_pos =y_rhb) And
             (x_rel_pos >= x_lht) And (x_rel_pos <= x_rhb) Then
          Begin found:= True; edge:= bottom End
          Else If (x_rel_pos =x_lht) And
             (y_rel_pos >= y_lht) And (y_rel_pos <= y_rhb) Then
          Begin found:= True; edge:= left End
          Else If (x_rel_pos =x_rhb) And
             (y_rel_pos >= y_lht) And (y_rel_pos <= y_rhb) Then
          Begin found:= True; edge:= right End
          Else If (x_rel_pos >= x_lht) And (x_rel_pos <= x_rhb) And
                  (y_rel_pos >= y_lht) And (y_rel_pos <= y_rhb) Then
               found := True
          Else
               t_e_pt := t_e_pt^.next_ent;
```

```
        {after this have either found entity or cursor isn't properly
         positioned}
        If Not found Then
        Begin
             Write('Error - entity not found, cursor may be badly');
             Write(' positioned')
        End;
        find_entity := t_e_pt
    End {find_entity};

    Procedure select_entity_for_repositioning;
    {Select the entity at the cursor position for moving}
    Var edge:rectedge;
    Begin
        {First find entity}
        move_ent := find_entity( edge );
        If move_ent <> Nil Then
        Begin
             {Move copy of entity box to work screen to enable
              moving to be animated}
             {first move cursor to lh corner of box for graphics}
             x_cur := move_ent^.ent.x_lht + xfac;
             y_cur := move_ent^.ent.y_lht + yfac;
             Writeln(grafix,'Q',x_cur,y_cur);
             Writeln(grafix,'X7');
             {First take window of entity on display screen}
             Writeln(grafix,'I',length+1,height+1);
             {Reposition cursor to centre of screen 1}
             {Move box and set window to it}
             Writeln(grafix,'V',1);
             Writeln(grafix,'I',length+1,height+1);
             {Go back to display screen}
             Writeln(grafix,'A',0);
        End {Else}
    End {select_entity_for_repositioning};

    Procedure reposition_entity;
    { To set new coords for entity & finish moving}
    Begin
        If move_ent = Nil Then
             Write('Error - No entity selected ')
        Else Begin
             With move_ent^.ent Do
             Begin
                  x_lht := x_cur - xfac;
                  y_lht := y_cur - yfac;
                  x_rhb := x_lht + length;
                  y_rhb := y_lht + height
             End;
             move_ent := Nil; {end of movement}
             Writeln(grafix,'A1');
             Writeln(grafix,'2'); {Clear work screen}
             Writeln(grafix,'A0');
        End
    End {reposition_entity};
```

```
Procedure add_new_ent;
{ To add a new entity at position pointed at}
Var nam_not_found: Boolean;
Begin
     {add to entity list & set up values}
     New( tlents^.next_ent );
     tlents := tlents^.next_ent;
     With tlents^.ent Do
     Begin
          Write('Enter entity name - ');
          Readln(ent_name);
          ent_name_lth := maxnamlth; nam_not_found := True;
          While (ent_name_lth>=1) And (nam_not_found) Do
               If ent_name[ent_name_lth] <> ' ' Then
                   nam_not_found :=False
               Else ent_name_lth := ent_name_lth - 1;
          x_lht := x_cur - xfac;
          y_lht := y_cur - yfac;
          x_rhb := x_lht + length;
          y_rhb := y_lht + height;
          {draw new entity box}
          Writeln(grafix,'X7');
          Writeln(grafix,'A0');
          write_entity_shape(x_lht+xfac,y_rhb+yfac,
                             x_rhb+xfac,y_lht+yfac);
          draw_name(ent_name,ent_name_lth,x_lht+xfac,y_lht+yfac);
          Writeln(grafix,'Q',x_cur,y_cur)
     End {With};
End {add_new_ent};

Procedure remove_ent( ent_pt: entpt);
{To remove the entity found}
Var t_e_pt: entpt;
Begin
     {Find previous entry & set pointers to bypass, should also
      dispose of it, but assume this isn't necessary for moment}
     {If is head of ents then just reset top pointer}
     If ent_pt = hdents Then
     Begin
          hdents := hdents^.next_ent;
          {If is only ent in list then set tl = nil}
          If tlents = ent_pt Then tlents := Nil
     End
     Else Begin
          t_e_pt := hdents;
          While (t_e_pt^.next_ent <> Nil) And
                (t_e_pt^.next_ent <> ent_pt) Do
                t_e_pt := t_e_pt^.next_ent;
          If t_e_pt^.next_ent = Nil Then
               Write('Error - corrupt rel pointers')
          Else Begin
            t_e_pt^.next_ent := t_e_pt^.next_ent^.next_ent;
            {move tlents if points at removed ent}
            If tlents = ent_pt Then tlents := t_e_pt
```

```
            End
        End {Else};
End{remove_ent};

Procedure delete_ent;
{ To remove an entity pointed at}
Var ent_pt: entpt;
    edge: rectedge;
Begin
     {First find rel concerned}
     ent_pt := find_entity( edge );
     If ent_pt <> Nil Then
     Begin
          Write('Confirm you wish to delete');
          get_command;
          If ch = 'T' Then
          Begin
               remove_ent( ent_pt );
               {remove ent from drawing by clearing area}
               Writeln(grafix,'X0');
               x_cur := ent_pt^.ent.x_lht+xfac;
               y_cur := ent_pt^.ent.y_lht+yfac;
               Writeln(grafix,'Q',x_cur,y_cur);
               Writeln(grafix,'I',length+1,height+1);
               Writeln(grafix,'V0')
          End
     End
End {delete_ent};

Procedure remove_rel( rel_pt: relpt);
{To remove the relationship found}
Var t_r_pt: relpt;
Begin
     {Find previous entry & set pointers to bypass, should also
      dispose of it, but assume this isn't necessary for moment}
     {If is head of rels then just reset top pointer}
     If rel_pt = hdrels Then
     Begin
          hdrels := hdrels^.next_rel;
          {If is only rel in list then set tl = nil}
          If tlrels = rel_pt Then tlrels := Nil
     End
     Else Begin
          t_r_pt := hdrels;
          While (t_r_pt^.next_rel <> Nil) And
                (t_r_pt^.next_rel <> rel_pt) Do
                t_r_pt := t_r_pt^.next_rel;
          If t_r_pt^.next_rel = Nil Then
               Write('Error - corrupt rel pointers')
          Else Begin
            t_r_pt^.next_rel := t_r_pt^.next_rel^.next_rel;
            {move tlrels if points at removed rel}
            If tlrels = rel_pt Then tlrels := t_r_pt
          End
     End {Else};
```

```
End{remove_rel};

Procedure delete_rel;
{ To delete a relationship pointed at by cursor }
Var rel_pt: relpt;
    pas:Boolean;
Begin
     {First find rel concerned}
     rel_pt := find_rel(pas);
     If rel_pt<>Nil Then
     Begin
          Write('Confirm you wish to delete');
          get_command;
          If ch = 'y' Then
          Begin
               remove_rel( rel_pt );
               {remove rel from drawing}
               Writeln(grafix,'X4');
               draw_a_relationship(rel_pt);
               {reposition cursor}
               Writeln(grafix,'Q',x_cur,y_cur)
          End
     End
End {delete_rel};

Procedure amend_rel;
{ To change a relationship in place}
Var pas: Boolean;
Begin
     new_rel_pt := find_rel( pas );
     If new_rel_pt <>Nil Then
     Begin
          start_amend_path := find_path( new_rel_pt ,
                                         (x_cur-xfac),
                                         (y_cur-yfac),
                                         no_of_paths );
          If start_amend_path = Nil Then
          Begin
               {something odd happened, stop change}
               new_rel_pt := Nil;
               no_of_paths := 0;
               Write('Error - error in paths')
          End Else Begin
               {move cursor to start of path}
               no_of_paths := no_of_paths - 1; {allow for dup. pth}
               x_cur := start_amend_path^.path.start_coord.x+xfac;
               y_cur := start_amend_path^.path.start_coord.y+yfac;
               Writeln(grafix,'Q',x_cur,y_cur);
               Writeln(grafix,'X7')
          End{Else}
     End{If}
End {amend_rel};

Procedure begin_rel;
{ To begin a relationship at entity pointed at by cursor }
```

```
var ent_pt: entpt;
    x_rel_pos, y_rel_pos: Integer;
Begin
     {First find entity being pointed at as rel must start
      from an entity}
     ent_pt := find_entity( rel_strt_edge );
     If ent_pt <>Nil Then
     If rel_strt_edge = niledge Then
          Write('Error - must point to an edge')
     Else Begin
          x_rel_pos := x_cur - xfac;
          y_rel_pos := y_cur - yfac;
          {Set up new rel entry}
          New ( new_rel_pt );
          With new_rel_pt↑.rel Do
          Begin
               {Set up default values}
               act_rel_opt := 'O';
               act_rel_deg := 'O';
               pas_rel_opt := 'M';
               pas_rel_deg := 'M';
               no_of_rels := 0
          End {with};
          no_of_paths := 0;
          new_rel_pt↑.paths := Nil; {paths are attached in
                                     start_new_path}
          new_rel_pt↑.next_rel := Nil;
          {rel is drawn by move_cursor }
          Writeln(grafix,'X7')
     End {Else}
End {begin_rel};

Procedure end_rel;
{ To end a relationship at entity arrived at}
Var ent_pt: entpt;
    edge: rectedge;
Begin
  If new_rel_pt = Nil Then
     Write('Error - no relationship selected')
  Else Begin
     ent_pt := find_entity( edge );
     If ent_pt <> Nil Then
     If edge = niledge Then
          Write('Error - must point to an edge')
     Else Begin
          With latest_path_pt↑ Do
          Begin
               next_path := Nil;
               path.end_edge := edge;
               path.end_coord.x := x_cur - xfac;
               path.end_coord.y := y_cur - yfac
          End {With};
          If start_amend_path <> Nil Then
          Begin
               {if amending want to be able to remove old path}
```

```
                    {remove old path on drawing}
                    Writeln(grafix,'X4');
                    draw_a_relationship(new_rel_pt);
                    {change start path of amend to be new set of paths}
                    start_amend_path↑ := first_new_path↑;
                    new_rel_pt↑.rel.no_of_rels := no_of_paths
               End
               Else Begin
                    {Set paths_pointer}
                    new_rel_pt↑.rel.no_of_rels := no_of_paths;
                    new_rel_pt↑.paths := first_new_path;
                    {remove path on drawing}
                    Writeln(grafix,'X4');
                    draw_a_relationship(new_rel_pt);
                    {add rel to tail of rel list}
                    tlrels ↑.next_rel := new_rel_pt;
                    tlrels := new_rel_pt
               End;
               {end of rel draw}
               Writeln(grafix,'X7');
               draw_a_relationship(new_rel_pt);
               Writeln(grafix,'Q',x_cur,y_cur);
               new_rel_pt := Nil;
               latest_path_pt := Nil;
               first_new_path := Nil;
               old_dir := ' '
          End {Else}
     End {Else}
End {end_rel};

Procedure change_rel_opt;
{ To change the optionality of the relationship end pointed at}
Var rel_pt: relpt;
    pas: Boolean;
Begin
     rel_pt := find_rel( pas );
     If rel_pt <> Nil Then
     With rel_pt↑.rel Do
     Begin
          If Not pas Then
          Begin
               {Is active end}
               If act_rel_opt = 'O' Then act_rel_opt := 'M'
               Else Begin
                    {remove old relationship}
                    Writeln(grafix,'X4');
                    draw_a_relationship( rel_pt );
                    act_rel_opt := 'O'
               End
          End
          Else If pas_rel_opt = 'O' Then pas_rel_opt := 'M'
          Else Begin
               {old relationship}
               Writeln(grafix,'X4');
               draw_a_relationship( rel_pt );
```

```
                pas_rel_opt := 'O'
            End;
            { redraw relationship }
            Writeln(grafix,'X7');
            draw_a_relationship( rel_pt );
            {reposition cursor}
            Writeln(grafix,'Q',x_cur,y_cur)
        End {with}
End {change_rel_opt};

Procedure change_rel_card;
{ To change the cardinality of the relationship end pointed at}
Var rel_pt: relpt;
    pas: Boolean;
Begin
     rel_pt := find_rel( pas );
     If rel_pt <>Nil Then
     With rel_pt^.rel Do
     Begin
          If Not pas Then
          Begin
               {Is active end}
               If act_rel_deg = 'O' Then act_rel_deg := 'M'
               Else Begin
                    {remove old relationship}
                    Writeln(grafix,'X4');
                    draw_a_relationship( rel_pt );
                    act_rel_deg := 'O'
               End
          End
          Else If pas_rel_deg = 'O' Then pas_rel_deg := 'M'
          Else Begin
                 {remove old relationship}
                 Writeln(grafix,'X4');
                 draw_a_relationship( rel_pt );
                 pas_rel_deg := 'O'
          End;
          { redraw relationship }
          Writeln(grafix,'X7');
          draw_a_relationship( rel_pt );
          {reposition cursor}
          Writeln(grafix,'Q',x_cur,y_cur)
     End {with}
End {change_rel_card};

Procedure start_new_path( dir : Char);
{To start a new path for a rel}
Var pth: pathpt;
Begin
     {Set up path values}
     New (pth);
     no_of_paths := no_of_paths + 1;
     With pth^.path Do
     Begin
          start_coord.x := x_cur - xfac;
```

```
                start_coord.y := y_cur - yfac;
                strt_edge := niledge;
                end_edge := niledge;
                If (dir = 'R') Or (dir = 'L') Then
                Begin
                     line_grad := 0;
                     line_const := start_coord.y Div yscale
                End Else Begin
                     line_grad := infinite_grad;
                     line_const := start_coord.x Div xscale
                End
           End {With};
           {Attach to path list or rel}
           If first_new_path = Nil Then
           Begin
                {first path, start list}
                first_new_path := pth;
                pth^.path.strt_edge := rel_strt_edge
           End
           Else Begin
                {finish last path}
                latest_path_pt^.path.end_coord := pth^.path.start_coord;
                latest_path_pt^.next_path := pth
           End;
           latest_path_pt := pth;
           Writeln(grafix,'X7');
      End {start_new_path};

      Procedure draw_new_line( dir: Char; num: Integer);
      {To draw a relationship and move cursor}
      Begin
           If (old_dir<>dir) And (old_dir <> 'A') Then
                {have either started a new line or changed direction
                 in either case is a new path and isn't start of amend}
                If (old_dir = ' ') Or
                   (((old_dir = 'R') Or (old_dir ='L')) And
                    ((dir = 'U') Or (dir = 'D'))) Or
                   (((old_dir = 'U') Or (old_dir = 'D')) And
                    ((dir = 'R') Or (dir = 'L')))  Then
                   {either new line or are orthogonal - need new path}
                   start_new_path( dir )
                Else Writeln(grafix,'X6');
           {now move cursor}
           With latest_path_pt^.path Do
           Case dir Of
                'R': Begin
                     Writeln(grafix,'f',num,0);
                     x_cur := x_cur + num;
                     End;
                'L': Begin
                     Writeln(grafix,'f',-num,0);
                     x_cur := x_cur - num;
                     End;
                'U': Begin
                     Writeln(grafix,'f',0,-num);
```

```
                    y_cur := y_cur - num;
                    End;
               'D': Begin
                    Writeln(grafix,'f',0,num);
                    y_cur := y_cur + num;
                    End
          End {Case};
     old_dir := dir;
End {draw_new_line};

Procedure move_cursor( dir:Char; num: Integer );
{ To move cursor & box as well if is a box move }
Begin
     If move_ent <> Nil Then
     Begin
          {Remove old entity at cursor}
          Writeln(grafix,'Al');
          Writeln( grafix,'X',6 );
          Writeln( grafix,'V',0)
     End;
     {If drawing rel then must do so}
     If new_rel_pt <> Nil Then draw_new_line( dir , num )
     Else
        {Move cursor to new position through relative positioning}
        Case dir Of
          'R': Begin
               Writeln(grafix,'R',num,0);
               x_cur := x_cur + num
               End;
          'L': Begin
               Writeln(grafix,'R',-num,0);
               x_cur := x_cur - num
               End;
          'U': Begin
               Writeln(grafix,'R',0,-num);
               y_cur := y_cur - num
               End;
          'D': Begin
               Writeln(grafix,'R',0,num);
               y_cur := y_cur + num
               End
        End {Case};

     If move_ent <> Nil Then
     {Place entity at new cursor position as is moving}
     Begin
          Writeln(grafix,'A',1);
          Writeln(grafix,'X',7);
          Writeln(grafix,'V',0)
     End
End {move_cursor};

Procedure write_interactive_menu;
Begin
     Writeln(grafix,'Q',0,0);
```

```
            Writeln(grafix,'iNORMAL');
            {First set complete band to reverse video}
            Writeln(grafix,'I',left_edge_after_menu,screen_bottom_edge);
            Writeln(grafix,'X15');
            Writeln(grafix,'V0');
            Writeln(grafix,'v');{Reverse Video}
            Writeln(grafix,'p',cr,'F1 Quit ',cr,
            'F2 Refresh',cr,
            '↑F2 Abandon',cr,
            'F3 Ent Move',cr,
            '↑F3 End Move',cr,
            'F4 New Ent',cr,
            '↑F4 Del Ent',cr,
            'F5 Del Rel',cr,
            '↑F5 Chge Rel',cr,
            'F6 Strt Rel',cr,
            '↑F6 End Rel',cr,
            'F7 Opt Chge',cr,
            '↑F7 Card Chg',cr,
            '           ',cr,
            'ESC Menu',cr,
            '    On/Off',cr,
            '           ',cr,
            'CONT - view',cr,
            '           ',cr,
            'CHANGE MODE'
            );
            Writeln(grafix,'w');
            Writeln(grafix,'iSMALL');
            Writeln(grafix,'Q',x_cur,y_cur);
      END {write_interactive_menu};

Begin
      write_interactive_menu;
      move_intialise;
      Repeat
            { interactive move command loop }
            get_command; {uses same keyboard so must reinterpret
                          same set of characters, numbers are
                          meaningless in this procedure }
            CASE ch OF
                  'K','k': move_cursor('L',1);
                  'J','j': move_cursor('R',1);
                  'I','i': move_cursor('U',1);
                  'M','m': move_cursor('D',1);
                  'S','s': move_cursor('L',3);
                  'A','a': move_cursor('R',3);
                  'W','w': move_cursor('U',3);
                  'Z','z': move_cursor('D',3);
                  'p'    : Begin {refresh screen}
                           box_on := False; draw_mod;
                           Writeln(grafix,'Q',x_cur,y_cur)
                           End;
                  'P'    : move_intialise;
                  'r'    : select_entity_for_repositioning;
```

```
                    'R'     : reposition_entity;
                    't'     : add_new_ent;
                    'T'     : delete_ent;
                    'y'     : delete_rel;
                    'Y'     : amend_rel;
                    'b'     : begin_rel;
                    'B'     : end_rel;
                    'v'     : change_rel_opt;
                    'V'     : change_rel_card;
                    'L','l': BEGIN
                         If menuon Then
                         Begin
                              menuon := False;
                              screen_left_edge := 0;
                              leftedge := 0
                         End Else Begin
                              write_interactive_menu;
                              screen_left_edge := left_edge_after_menu;
                              leftedge := screen_left_edge;
                              CH := ' ';
                              menuon := True;
                         END;
                    END;
                    'G','g': { end interactive mode };
                    'Q' ,'q':(* QUIT*);
                    ' ','C','c','N','n','E','e','U','u',
                     'X','x','D','d','F','f','H','h': CH := ' '
               END (* CASE *);

          UNTIL (CH = 'Q') Or (ch = 'q')
               Or (ch = 'g') Or (ch = 'G');
          {Write view menu if not quitted}
          If (ch <> 'Q') And (ch <> 'q') Then
          Begin
               {Set up things for return from mode}
               If menuon Then writemenu;
               box_on := False;
               draw_mod
          End;
          Writeln(grafix,'r'); {disable cursor}
     End {interactive_move_mode};


BEGIN
     initialise;
     REPEAT
          get_command;
          CASE ch OF
               'K','k': move_drawing('R',curr_y_dir,no_of_times);
               'J','j': move_drawing('L',curr_y_dir,no_of_times);
               'I','i': move_drawing(curr_x_dir, 'U',no_of_times);
               'M','m': move_drawing(curr_x_dir,'D',no_of_times);
               'R','r': BEGIN
                    XFAC := screen_left_edge+(length div 2);
                    YFAC := YCENTRE;
```

```
Begin
     read_stuff;
     draw_plot
End.
```

```
                curr_x_dir := ' ';
                curr_y_dir := ' ';
                SPEED := 0;
                box_on := False;
                draw_mod
            END (* 'R' *);
            'W','w': move_drawing(' ','U', (no_of_times*(ycentre div 2)));
            'S','s': move_drawing('R',' ', (no_of_times*(xcentre div 2)));
            'A','a': move_drawing('L',' ', (no_of_times*(xcentre div 2)));
            'Z','z': move_drawing(' ','D', (no_of_times*(ycentre div 2)));
            'B','b':change_scale(no_of_times,no_of_times);
            'V','v':change_scale(-no_of_times,-no_of_times);
            'P','p': Writeln(grafix,'?');
            'L','l': BEGIN
                If menuon Then
                Begin
                      {set toggle and redraw model under menu}
                      menuon := False;
                      screen_left_edge := 0;
                      leftedge := 0;
                      rightedge := left_edge_after_menu;
                      draw_mod;
                      rightedge := screen_right_edge
                End Else Begin
                     WRITEMENU;
                     screen_left_edge := left_edge_after_menu;
                     leftedge := screen_left_edge;
                     CH := ' ';
                     menuon := True;
                END;
            END;
            'Y','y':Begin optality_on := False;
                          Writeln(grafix, 'Z1' );
                          draw_mod
                     End;
            'T','t': Begin optality_on := True;
                           opt_set := False; draw_mod
                     End;
            'G','g': interactive_move_mode;
            'Q' ,'q':(* QUIT*);
            ' ','C','c','N','n','E','e','U','u',
             'X','x','D','d','F','f','H','h': CH := ' '
           END (* CASE *);

           If (ch = 'A') Or (ch = 'a') Or (ch = 'S') Or (ch = 's') Or
              (ch = 'z') Or (ch = 'Z') Or (ch = 'W') Or (ch = 'w') Or
              (ch = ' ') Then
           Begin
                curr_x_dir := ' '; curr_y_dir := ' '
           End;
    UNTIL (CH = 'Q') Or (ch = 'q');
    Writeln(escape,'m2',' ','8');
    Writeln(escape,'E');
END (* draw_plot *);
```

APPENDIX X7 - INITIAL PROPOSAL

The following is a copy of the intial proposal of the research as accepted by the SERC.

After this is a copy of the initial proposal as accepted by the CNAA.

SRC Science Research Council
PO Box 18
Swindon SN2 1ET

CO-OPERATIVE AWARDS IN SCIENCE AND ENGINEERING (INCLUDING RESEARCH COUNCILS CO-OPERATIVE AWARDS) APPLICATION FOR THE APPROVAL OF A PROJECT

RS1A–1981

1. THE FORM MUST BE COMPLETED BY THE HEAD OF DEPARTMENT AND THE ORIGINAL PLUS 6 COPIES SENT TO SRC. THERE IS NO CLOSING DATE FOR SUBMISSION OF THE FORM, WHICH MAY BE SUBMITTED AT ANY TIME.
2. Applications from other Research Council Institutes/Units acting as the academic or collaborating body will be accepted, but will be regarded as requests for Research Council Co-operative Awards (RCCA).
3. BEFORE COMPLETION PLEASE READ THE NOTES ON PAGE 4 OF THE FORM.
4. The form must be completed in typescript or black ink. Enter a single letter or figure in each space, where a box is provided. Numerical entries should be made on the right entering zeros in any blank spaces remaining to the left.

Document Code 2
Transaction Code Q 5 2

FOR SRC USE

13 CBC Code | Institution Code | Department Code | Clerical Number

SECTION I

1. Institution: THAMES POLYTECHNIC

2. Department: MATHEMATICS, STATISTICS & COMPUTING

3. Applicant (Academic Supervisor) Title: MR | Initials: G | Surname: FITZGERALD

Telephone No. 01-854 2030 Extension 377

4. Is the institution at 1. above a unit or institute of another Research Council? (YES or NO) ....NO....

If YES, please state which Council .....

5. Project reference 53

FOR SRC USE

	Acc. Date
To consider	Queries
Approved/Rejected	Initials Date
Announced (Date)	Allocated No.

6. Number of Studentships requested. (If more than one, reasons should be given in the description of the project). [1]

7. Have you submitted a student nomination with this project? Put an X in box if YES [ ]

If YES, student's name .....

8. Publicity Marker. Put an X in this box if you or the co-operating body wish the project to be excluded from a list circulated to Institutions [ ]

Transaction Code Q 5 3

9. Name of Co-operating Body (Not more than 30 characters including spaces)

13 CACI, INC - INTERNATIONAL

10. Address of Co-operating Body

289 High Holborn
London WC1V 7HZ

11. Co-operating Body's supervisor or contact, and address if different from that of the Co-operating Body

Mr. A.A.Shareef

12. If Co-operating Body is a unit/institute of another Research Council, please state which Council:

NO

Telephone No. 01-405 8233 Extension

13. Brief title of project (Not more than 60 characters including spaces) Transaction Code Q 5 4

Note: This entry will be used in computer-produced lists, including publicity where appropriate.

13 DATA ANALYSIS SUPPORT SYSTEM

14. Have you ever applied for or received an SRC Research Grant for the area of research involved in this project (YES or NO)....NO....

If 'YES' please give Grant Reference No. .....

15. Please state whether or not the proposed CASE/RCCA co-operation has any connection with any other programme of work where a third party has rights. (YES or NO) .....NO.....

(If the proposed programme falls within an area of work for which there has been e.g. an SRC grant, property rights deriving from the programme might be dominated by rights deriving from the grant, the latter being assigned (or assignable) to NRDC).

FOR SRC USE 79 Transaction Code

Source

Title of project	Investigation and extension of data analysis methodology by development of a support system	FOR SRC USE Ref. No.

Use of SRC Central Facilities

To help SRC in forward planning, give details of the access students will require to:

i) SRC supported Central Facilities in the UK NONE

ii) Overseas Facilities NONE

N.B. THE AWARD OF STUDENTSHIPS DOES NOT NECESSARILY MEAN THAT ACCESS TO THESE FACILITIES WILL BE GRANTED.

A. Description of project (to be completed by the Academic Supervisor, see Note 4). Please continue on separate sheet if necessary.

Please see attached document

B. Nature of Co-operation (To be completed by the Co-operating Body's Supervisor, see Note 5).

(a) Description of the work likely to be undertaken by the student at the premises of the Co-operating Body.

The student will initially be trained by CACI in data analysis skills and will have access to teams working in this area. During the project he will spend 3 periods with CACI and work on projects, under supervision, to enable him to relate his research work to the pragmatic nature of the work done by CACI.

(b) Student's contact with the Co-operating Body Indicate approximate total time spent with collaborating body and the duration and frequency of visits.

Three periods of one month each, plus visits as appropriate. An initial training period with CACI is envisaged.

(c) (i) Academic Supervisor's contact with the Co-operating Body.

Three visits per year

(ii) Industrial Supervisor's contact with Institution.

Three visits per year

Financial contribution by the Co-operating Body It is very important to read Note 5 overleaf.

(d) (i) Students' travel and subsistence.

£100

(ii) Annual cash contribution to department.

£400

(iii) Additional contribution (description and value)

£850 (training and documentation)

(e) Payment to student

£500

SECTION III (Certificates)

(i) For the Institution Administrative Authority
This application is made with the knowledge and approval of the Institution.

Signature of Head of Department .....

Date 080981

Signature of Administrative Authority .....

Date 8/9/81

(ii) For the Co-operating Body

I hereby confirm that should a studentship be awarded for the project as described in Section IIA, support for the project as detailed in Section IIB will be provided.

Signed on behalf of the Co-operating Body .....

Position Held Vice President

Date 15/7/81

This page to be retained by applicant

CASE AWARDS – PATENTS OR OTHER EXPLOITABLE RESULTS GUIDE LINES FOR UNIVERSITIES (OR OTHER INSTITUTIONS)

1. The programme of research should be defined as clearly as possible together with the objectives. This will help to distinguish the work and its results from any other programme of work that might be carried out in the institution in the same or similar fields.

2. The respective contributions of the SRC, the co-operating body and the other parties to the research should be stated and, when possible, valued in money terms over the duration of the award. SRC's contribution will take into account the following, as appropriate; Basic award, allowances, travel and subsistence payments, and any other payments to the student; tuition, college and student union fees, Research Training Support Grant and any other payments to the institution; administration costs at 5% of the total of the above.

3. It is reasonable for the co-operating body to insist that any information regarding its operations (including technical and commercial information) that is acquired by the other parties to the research shall not be used (except in connection with the work), published or otherwise disclosed, without the co-operating body's permission.

4. None of the parties to the research should be free to publish or otherwise disclose the results of the programme without the consent of the others, such consent not to be unreasonably withheld.

5. In order to satisfy the interests of the co-operating body the SRC is prepared to waive any claim it may have to an interest in results obtained during the course of the studentship, provided satisfactory terms can be agreed between the supervisor and/or student and/or institution and the co-operating body. The SRC waiver does not (and cannot) extend to the rights of any others who might be co-inventors with the student.

6. When the exploitable results are to be acquired by the co-operating body, it should be in return for a royalty payable to the other parties to the research and based on the commercial use of the results by the co-operating body. When a patent is involved the royalty should be for the life of the patent. In the case of know-how (no patent) it is reasonable that payments by the co-operating body should cease after a fixed negotiated period (e.g. 7 years from first marketing).

7. Preferably there should be provision for the co-operating body to relinquish its rights to the other parties to the research or to NRDC if, after a reasonable period, the co-operating body has not exploited these rights commercially. Relinquishment would usually be subject to a free licence back to the co-operating body if needed.

8. If, having acquired the exploitable results, the co-operating body subsequently decides not to use them commercially, there should be provision for the results to be offered to the other parties to the research or to NRDC in good time to enable patent or other protection to be maintained.

9. When the results are acquired by the co-operating body, it is to be preferred that the other parties to the research or NRDC should have right to a licence (with rights to sub-license) to the extent that such a licence might be needed in order to exploit some other results belonging to the institution, to the SRC, or to NRDC.

10. The agreement should provide, where the co-operating body does not wish to acquire the results, for those results to be offered to NRDC by the other parties to the research.

11. Any party concerned in a CASE Award may refer the agreement to NRDC for comment in order to ensure that the interests of all the parties are protected. It is emphasised, however, that NRDC (except in the circumstances mentioned below) is not a principal and that it is the responsibility of the other parties to the research to negotiate with the co-operating body and to arrive at an arrangement that satisfactorily takes account of the contributions to the project, including the SRC contribution.

12. The NRDC can appear as a principal in the negotiation when the institution or co-operating body involved in the CASE Award is one whose inventions are normally offered to the Corporation for exploitation.

NOTE: THE ABOVE GUIDELINES APPLY ONLY TO CASE AWARDS. FOR RESEARCH COUNCILS CO-OPERATIVE AWARDS SEE PAGE 4, PARA 8.

THIS PAGE SHOULD NOT BE PHOTOCOPIED

CASE/RCCA STUDENTSHIPS

REPLY SLIP

SCIENCE RESEARCH COUNCIL
PO Box 18, Swindon SN2 1ET
Tel: 0793-26222 Ext. 2138

Project title: Data Analysis Support System

Guy Fitzgerald
Thames Polytechnic
Wellington Street,
London SE18 6PF

Name and address of Applicant (Academic Supervisor)

The application for the approval of this project has been considered and the result is enclosed.

Please quote the following reference in any correspondence

Reference Number

ACKNOWLEDGEMENT SLIP
CASE/RCCA STUDENTSHIPS
If not received within two weeks please notify SRC

SCIENCE RESEARCH COUNCIL
PO Box 18, Swindon SN2 1ET
Tel: 0793-26222 Ext. 2138

Academic Supervisor: Guy Fitzgerald

Project title: Data Analysis Support System

Dr. J.M. Hobbs,
Thames Polytechnic
Wellington Street,
London SE18 6PF

Name and address of Head of Department

The application for the approval of this project has been received

Please quote the following reference in any correspondence

Reference Number

Please pass/show this acknowledgement to the Academic Supervisor.

SRC RECORD SLIP (Please complete this slip to ensure SRC has accurate records of projects etc.)

FOR SRC USE

ACADEMIC SUPERVISOR(S): GUY FITZGERALD

INSTITUTION:
ames Polytechnic

DEPARTMENT:
thematics,
atistics and

CO-OPERATING BODY:
CACI, INC INTERNATIONAL
289 High Holborn,
London WC1V 7HZ

PROJECT TITLE:
Data Analysis
Support System

FOR SRC USE
Approv'd/Reject'd | Initials + Date
Announced (Date)
STUDENT DETAILS
1
2

BEST COPY AVAILABLE.

TEXT IN ORIGINAL IS BOUND INTO THE SPINE

CASE/RCCA STUDENTSHIPS

Please complete the following slips to minimise delay in announcing results

REPLY SLIP

SCIENCE RESEARCH COUNCIL
PO Box 18, Swindon SN2 1ET
Tel: 0793-26222 Ext. 2138

Institution Thames Polytechnic
Academic Supervisor Guy Fitzgerald

Project title: Data Analysis Support System

Mr. A.A. Shareef
CACI, Inc-INternational
289 High Holborn
London WC1V 7HZ

Name and address of Co-operating Body's contact

The application for the approval of this project has been considered and the result is enclosed.

Please quote the following reference in any correspondence

Reference Number

CASE/RCCA STUDENTSHIPS

REPLY SLIP

SCIENCE RESEARCH COUNCIL
PO Box 18, Swindon SN2 1ET
Tel: 0793-26222 Ext. 2138

Applicant (Academic Supervisor) GUY FITZGERALD

Project title: Data Analysis Support System

Mr. J.M. Hobbs,
Thames Polytechnic
Wellington Street
London SE18 6PF

Name and address of Head of Department

The application for the approval of this project has been considered and the result is enclosed.

Please quote the following reference in any correspondence

Reference Number

CASE/RCCA STUDENTSHIPS

REPLY SLIP

SCIENCE RESEARCH COUNCIL
PO Box 18, Swindon SN2 1ET
Tel: 0793-26222 Ext. 2138

Department: Mathematics, Statistics and Computing

Academic Supervisor: Guy Fitzgerald

Project title: Data Analysis Support System

Academic Registrar,
Thames Polytechnic,
Wellington Street
London SE18 6PF

Name and address of Institution's Administrative Authority

The application for the approval of this project has been considered and the result is enclosed.

Please quote the following reference in any correspondence

Reference Number

DATA ANALYSIS SUPPORT SYSTEM

BACKGROUND

In recent years there have been a number of important developments in the field of systems analysis and database design. These have stemmed from the development of database technology and the resulting separation of data from programs which has been permitted, Martin (1975), Palmer (1975), Date (1977). This led to the database approach to systems analysis. However it was soon realized that this approach was limited by the prevailing state of database technology, and that for systems analysis purposes this was highly restrictive and needed to be abandoned. As a result the wider concept of data analysis was born.

The primary purpose of data analysis is to determine the fundamental data resources of an organization irrespective of any file structure, database, or implementation considerations. One of the most important aspects of this process being the identification of entities, and the associated relationships between entities. The resulting entity model reflects the fundamental data resources and their inter relationships. This

graphical representation provides a vital communication tool and has been found in practice to be of immense value as a succinct description of the enterprise understandable by both user and analyst.

The concept of data analysis has been developed by a number of organizations, most notably CACI (Palmer 1978). They have successfully used the technique for over five years in which time the initial concept has been developed into a complete analysis and design methodology.

PROPOSAL

The methodology has proved very successful in practice and has been consistently refined and developed by CACI. As a further stage in this process it is proposed that on the basis of a collaborative venture and via an SRC CASE award the methodology be examined in three areas:

1. Data modelling and model equivalences

Recent work particularly in the area of data modelling

and the equivalence of models indicates that futher formal techniques may be applicable in the area of data analysis. (E.g. Borkin (1980)). This work is based on the idea that different users having different data models might want to use the 'same' database. Using and developing these ideas it is proposed to develop techniques to establish the equivalence of conceptual models. The potential then exists to project equivalent models to the logical and physical levels. This will enable a number of advances, for example the ability to be able to establish rules for system portability and even for the generation of code.

2. Automation of data analysis processes

It is recognized that some of the techniques performed in the data analysis process are susceptible to automation. For example the process of generating relational schemas from a given set of functional dependencies has been shown to be an algorithmic process (Bernstein et al (1975)). In addition the process of developing the entity model and representing it on a screen would be possible and of great practical benefit, particularly in improving the user interface. It would involve optimally laying out the model to minimise the length of the lines denoting relationship types and the number of lines that cross. The entire model is unlikely to fit on a screen in a practical application so that it should be possible to scroll up and down and left and right.

3. Requirements for distributed systems

Techniques for introducing the requirements of distributed systems into the data analysis process need to be developed. For example the elements that need to be supported for distributed systems in both the functional and data areas, and their local variations and possible duplication.

The three areas discussed above are linked by their relationship to the data analysis methodology. The research vehicle will be the development of a software system supporting the three areas. The development of this work to aid the data analysis process would be of emmense practical value as well as a substantial academic task.

OBJECTIVES

To formalize this the project has as its main objectives:

1. The establishment of a full understanding of the data analysis concept and the CACI data analysis methodology.

2. The development of a software system to provide support for the data analysis process.

This will require exploration of the following areas:

a) Data modelling and the equivalence of conceptual models

b) Graphics and graph theory

c) Functional dependencies and algorithmic processes for generating normal form relations

d) Optimisation and evaluation of the generated normal form relations

e) The analysis of factors effecting distribution of data and processes, possibly using formal techniques e.g. cluster analysis

EVALUATION

The project is regarded as containing significant academic and scientific merit to justify its consideration as a research undertaking and is in the mainstream of current developments in data processing and systems analysis. The project offers structured development as research training and has enough potential to justify a full research studentship leading to a PhD. The co-operative nature of the project is regarded as a major strength and an important factor for its success. The co-operating body, CACI, is highly committed to the development of the methodology of data analysis and will ensure the practical success of the project and provide the necessary environment for

experimentation and testing. It is envisaged that the project will be developed at Thames Polytechnic with the results of each stage being tested, validated, and refined by periods of practical involvement at CACI.

CACI has previously worked closely with Universities and Polytechnics (Shave, Davenport, etc) and has a reputation for its own academic work (Palmer, Baker, etc). CACI will participate on both the practical and theoretical aspects of the research.

BERNSTEIN, P.A. SWENSON, J.R. TSICHRITZIS, D.C. (1975) A Unified Approach to Functional Dependencies and Relations, ACM-SIGMOD 1975.

CHEN, P.P.S. (1976) The entity-relationship model - Towards a unified view of data, ACM Transactions on Database Systems 1,1.

CODD, E.F. (1970) A relational model of data for large shared data banks, Communications ACM 13,6.

DAVENPORT, R.A. (1978) Data Analysis for Database Design, Australian Computer Journal, 10,4.

PALMER, I.R. (1975) Database Systems - A Practical Reference, CACI.

PALMER, I.R. (1978) Practicalities in applying a formal methodology to data analysis, Proceedings of NYU Symposium on Database Design.

FORM CNAA 9(R)
(1978)

For Official Use
Date Rec.
Item No.

COUNCIL FOR NATIONAL ACADEMIC AWARDS

(Attention is drawn to the Notes given on the attachment to this form)

(This application should be typewritten)

Application to register for the Council's Research Degree

Submitted by .....Thames Polytechnic.....
(Sponsoring Establishment)

for the degree of: (i) [redacted]
(ii) *Master of Philosophy with transfer possibility to Doctor of Philosophy (MPhil/PhD)
(iii) [redacted]
**delete as appropriate*

1. The Applicant

(Male/Female*)
**delete as appropriate*

Name: Paul Barrie FELDMAN B.Sc.(Hons) AMBCS

Date of Birth: 5.4.59.

Private address: 13 Queens Road,
Ealing,
London, W5 2SA.

Present post and place of work:
Research Student
School of Mathematics,
Statistics & Computing,
Thames Polytechnic

Particulars of any scholarship or other award held in connection with the proposed research programme:
SERC CASE AWARD

Qualifications gained (Regulation 2 refers) *(include place(s) of higher education, courses completed, main subjects, classification of award, date and name of awarding body)*:

B.Sc.(Hons) 1st Class, University of St.Andrews in Computational Science, 1981
1st Year App + Methods Maths, Physics, Astronomy.
2nd Year App + Methods Maths, Computational Science.
Jun.Hon. Computational Science
Sen.Hon. Computational Science

Training and experience *(include details of activities (with dates) relevant to this application, and of any research or other relevant papers, books, etc., which have been published)*:

Computer Programmer - Scicon UK Ltd. from 3.8.81 - 31.8.82.
GEC Hirst Research Centre from 1.7.80 - 31.9.80.

2. Academic Referees—see Note (a)

Not Applicable

3. Name of Collaborating Establishment—see Note (b)

CACI, INC - International. CACI were instrumental in defining the area of work and are active participants in the programme. CACI expect the results to be of great benefit to them in their future work.

4. The Programme of Research

4.1 Title of the proposed investigation

A theoretical and practical study of data analysis and data models.

4.2 Aim of the investigation:

To develop the theoretical understanding of data analysis and to enhance its practical use by means of a software system of support.

4.3 Proposed plan of work, including its relationship to previous work, with references (See Note (c)):

See further information attached.

4.4 Details of facilities available for the investigation (including funding and location):

Thames Polytechnic library and computing facilities. The computing facilities are based on a network of PRIME 750s plus DEC LSI 11s and extensive microcomputer systems.

CACI, Inc-International library and computing facilities.

4.5 Relationship between work to be undertaken in the collaborating establishment and that to be undertaken at the sponsoring establishment or elsewhere (Regulation 3.6 refers):

The candidate's work will be primarily undertaken at Thames Polytechnic. The candidate's work at the collaborating establishment will be to validate and test work done at the sponsoring establishment and to make use of computing facilities. CACI have significant areas of common interest (See 4.3) and are currently developing systems in related areas themselves (ADAM/MADAM 1982).

5. The Programme of Related Studies (Complete *either* 5.1 *or* 5.2)

5.1 Details of programme of related studies to be undertaken (Regulations 3.8 and 3.9 refer):

See further information attached.

5.2 Where an integrated programme of study is proposed, details of the course of postgraduate study on which candidate's performance is to be formally assessed (Regulation 3.10 refers) are required:

Not Applicable

6. Supervision of Programme of Work (Regulation 6 refers)

6.1 Director of Studies *(see Note d) (include name, qualifications, post held and place of work)*:

G.Fitzgerald, B.A., M.Sc., MBCS.
Senior Lecturer,
School of Mathematics, Statistics and Computing,
Thames Polytechnic

(See attached C.V)

Experience of supervision of *registered* research degree candidates:

Currently supervising ....1.... *CNAA candidates and* ....0.... *other candidates*

Previously supervised ....1.... *CNAA candidates and* ....0.... *other candidates*

(Successfully completed supervision)

6.2 Second Supervisor(s) *(see Note d) (include name, qualifications, post held and place of work)*:

T.Crowe, B.Sc., MBCS,
Head of Division,
Thames Polytechnic.

B.Knight, B.Sc., Ph.D.,
Senior Lecturer,
Thames Polytechnic.

(See attached C.Vs)

Experience of supervision of *registered* research degree candidates:

Currently supervising ....1....0.... *CNAA candidates and* ....0....0.... *other candidates*

Previously supervised ....1....0.... *CNAA candidates and* ....0....0.... *other candidates*

(Successfully completed supervision)

6.3 Details of any other person(s) who will act in an advisory capacity *(name, qualifications, post held and place of employment)*:

T.Bourne, B.Sc., B.A., MBCS,
Principal Consultant,
CACI, Inc-International,
High Holborn,
London, WC2.

7. **Period of Time for Completion of Programme of Work** (Regulation 4 refers)

7.1 Expected starting date for registration purposes (Appendix 1 of the Regulations refers) ....1st October, 1982.....

7.2 Mode of study (full-time or part-time)..... Full-time .....

7.3 Amount of time (hours per week average) allowed for programme .....40.....

Expected duration of programme (in years) on the above basis to MPhil .....1..... and additionally to PhD ...2.....

8. **Statement by the Applicant**

I wish to apply for registration for .....M.Phil/Ph.D..... on the basis of the proposals given in this application.

I confirm that the particulars given in Section 1 are correct.

I understand that, except with the specific permission of the Council, I may not, during the period of my registration, be a candidate for another award of the CNAA or of a University.

I understand that, except with the specific permission of the Council, I must prepare and defend my thesis in English.

Signature of applicant ..... Date .....12th January 1983.....

9. **Recommendation by the Supervisors**

We support this application and believe that .....Paul Barrie Feldman..... has the potential to complete successfully the programme of work proposed.

We recommend that this applicant be registered as a candidate for the Council's research degree.

Signed ..... Date .....18/1/83.....

Signed ..... Date .....18th Jan 83.....

10. **Recommendation by the Sponsoring Establishment** (*unless Section 11 below is completed*)

I support this application for registration of ..... as a candidate for a research degree of the Council

Signed ..... Date .....

Signature of Head/Principal Officer of the Establishment or authorised deputy or Chairman/Secretary of approved college Research Degrees Committee)

11. **Notification of Registration on behalf of the Council**

Note: This section may be completed only by a college which has a Research Degrees Committee approved by the Council.

I confirm that the candidate was registered by this college for the degree of MPhil or MPhil/PhD.*

on ..... with effect from .....

Signature ..... Date .....

Chairman/Secretary of approved college Research Degrees Committee **delete as appropriate*

The sponsoring establishment should send the completed Form and any attachments together with the appropriate application fee to the Council's Registrar for Research Degrees at: 344–354 Gray's Inn Road, London WC1X 8BP.

4.3 Proposed plan of work

Introduction

In recent years there have been a number of important developments in the field of systems analysis and database design. These have stemmed from the development of database technology and the resulting separation of data from programs which has been permitted (Pamler 1975, Date 1977). This led to the database approach to systems analysis. However it was soon realised that this approach was limited by the prevailing state of database technology, and that for systems analysis purposes this was highly restrictive and needed to be abandoned. As a result the wider concept of data analysis was born.

The primary purpose of data analysis is to determine the fundamental data resources of an organisation irrespective of any file structure, database, or implementation considerations. One of the most important aspects of this process being the identification of entities, and the associated relationships between entities. The resulting entity model reflects the fundamental data resources and their inter-relationships.

The concept of data analysis has been developed by a number of organisations, most notably CACI (Palmer 1978). They have successfully used the technique for a number of years in which time the initial concept has developed considerably (Rock-Evans 1981). Elsewhere similar activities have been pursued (Chen 1980, Flavin 1981, Shave 1981, Davenport 1979).

Proposal

Although data analysis has proved very useful in practice and is widely recognised to be a major advance it is still in the early stages of its development. It is argued that there are areas where the theoretical foundations need to be examined and established and that there are a number of potential avenues of development as yet uninvestigated.

These areas are as follows:

1. Data models and model equivalences

   Recent work in the area of the equivalence of models indicates that further formal techniques may be applicable in the area of data analysis, (Borkin 1980). This work is based on the idea that different users having different data models might want to use the 'same' database. Using and developing these ideas it is proposed to develop techniques to establish the equivalence of conceptual models. The potential then exists to project equivalent models to the logical and physical levels. This will enable a number of advances, for example the ability to be able to establish rules for system portability and for the generation of code.

2. Automation of data analysis processes

   It is recognised that some of the techniques performed in the data analysis process are susceptible to automation. For example the process of generati relational schemas from a given set of functional dependencies has been shown to be an algorithmic process (Bernstein et al 1975). In addition the process of developing the entity model and representing it on a screen would be possible and of great practical benefit, particularly in improving the user interface. It would involve optimally laying out the model to minimise the length of the lines denoting relationship types and the number of lines that cross. The entire model is unlikely to fit on a screen in a practical application so that it should be possible to scroll u and down and left and right.

Continued..../2

3. Requirements for distributed systems

Techniques for introducing the requirements of distributed systems into the data analysis process need to be developed. For example the elements that need to be supported for distributed systems in both the functional and data areas, and their local variations and possible duplication.

Programme

Initially an extensive review of data analysis concepts and methodologies will be undertaken by the candidate. This will include reference to related methodologies based on other models (De Marco 1979), Olle 1982 etc).

Following the review a simple data analysis data capture and retrieval system will be developed. This will allow entities and functions to be input and output. Subsequently a VDU graphics model will be developed to use and verify analysis results. This will enable an entity model to be displayed and manipulated. This work will involve investigation of existing graphics techniques (e.g. Newman 1981). The possibility of interfacing this work with other systems performing ancillary functions will be investigated (e.g. ADAM/MADAM 1982).

In parallel to the above work, and then subsequently using the software as a research tool, the concepts of data model equivalence will be investigated and its usefulness and practicality will be evaluated.

In addition the information required for making decisions concerning the distribution of data will be investigated. It is anticipated that this will have implications for the process of data analysis and that new information will need to be captured.

Whilst data analysis is now a commonly used technique the above proposals are believed to be novel.

Possible Ph.D. transfer

Upon completion of the above work the following areas of academic study will be considered:

1. The full development of methods for analysing data model equivalences.

2. The design/modification of a data analysis methodology to handle the requirements of distributed systems.

3. The further development of a software system to support the theoretical developments in data analysis.

It is anticipated that many other areas of interest will emerge as the research progresses and that these may become candidates for further research.

References

1. ADAM/MADAM (1982)-CACI internal report.

2. BERNSTEIN, P.A., SWENSON, J.R., TSICHRITZIS, D.C. (1975) - A Unified Approach to Functional Dependencies and Relations, ACM-Sigmod 1975.

3. BORKIN, S. (1980) - Data models: A Semantic approach for Database Systems, MIT Press.

4. CHEN, P.P.S. (1976) - The entity-relationship model - Towards a unified view of data, ACM Transactions on Database Systems 1,1.

5. CHEN, P.P.S. (1980) editor - The entity-relationship approach to Systems Analysis and Design, North-Holland.

Continued..../3

6. CODD, E.F. (1970) - A relational model of data for large shared data banks, Communications ACM 13,6.

7. DAVENPORT, R.A. (1978) - Data Analysis for Database Design, Australian Computer Journal, 10,4.

8. DE MARCO, T. (1979) - Structured Analysis and Systems Specification, Prentice Hall/Yourden Press.

9. NEWMAN, W. (1981) - Principles of interactive computer graphics, McGraw-Hill.

10. OLLE, T.W. (1982) Editor - Information Systems Methodologies: A comparati review, North-Holland.

11. PALMER, I.R. (1975) - Database Systems - A Practical Reference, CACI.

12. PALMER, I.R. (1978) - Practicalities in applying a formal methodology to data analysis, Proceedings of NYU Symposium on databases.

13. SHAVE, M.J.R. (1981) - Entities, functions and binary relations, The Computer Journal Vol.24, No.1, Feb.1981.

5.1 Details of programme of related studies to be undertaken

The candidate has only recently (1981) completed a four year degree course in computational science and does not therefore require another extensive course of training in the general subject area. However certain specific gaps need to be filled and a programme of conferences, seminars, courses and reading has been arranged. This is as follows:

Conferences

November 1982, Data Dictionaries. Arranged by the British Computer Society Data Dictionary Working Party.

December 1982, Distributed Databases. Arranged by the British Computer Society Distributed Database Working Party.

Seminars

November 1982, January 1983 - BCS Information Systems Analysis and Design Working Party seminars.

Thames Polytechnic School of Mathematics, Statistics and Computing internal seminars.

Various BCS Database specialist Group seminars.

Courses

October to December 1982, Databases. Thames Polytechnic Computing Science Honours Degree option.

September 1982, Data Analysis. One week full-time course run by CACI, Inc.-International.

Reading

Books

CHEN, P.P. (1980) Ed. - Entity-Relationship Approach to Systems Analysis and Design, North-Holland.

NEWMAN, W., SPROULL, R. (1981) - Principles of Interactive Computer Graphics, McGraw-Hill.

WAITE, M. (1979) - Computer Graphics Primer, H.W.Sams & Co.Inc.

ROCK-EVANS, R. (1981) - Data Analysis,Computer Weekly/IPC Business Pro.

DE MARCO, T. (1979) - Structured Analysis and System Specification, Prentice-Hall/Yourdon.

BORKIN, S. (1980) - Data Models: A Semantic Approach for Database Systems, MIT Press.

DRAFFAN, I. and POOLE, F. (1980) - Distributed Databases - An Advanced Course, Cambridge University Press.

WILSON, R. (1979) - Introduction to Graph Theory, Longman.

MARTIN, J. (1981) - Design and Strategy for Distributed Data Processing, Prentice-Hall.

FLAVIN, M. (1981) - Fundamental Concepts of Information Modelling, Yourdon Press Monograph.

Continued..../2

TSKARITZIS, D. and LOCHOUSKY, F. (1982) - Data Models, Prentice-Hall.

KENT, W. (1978) - Data and Reality, North Holland.

Papers

BCS DDSWP (1977) - Data Dictionary Systems Working Party Report.

E.F.Codd — A Relational Model of Data for large Shared Data Banks - Communications of the ACM Vol.13 (June

" — Further Normalisation of the Relational Model - Database Systems, Courant Computer Science Symposium 6, Prentice Hall 1972 (Ed.Rustin).

" — Relational Completeness of data sublanguages - As above.

" — A Database Sublanguage Founded on the relational Calculus - RJ 893, July 71.

R.A.Davenport — Data Analysis - Experience with a formal methodology - IFIP 79, North Holland, Ed.Samet.

I.Palmer — - Also in above

M.J.R.Shave — Entities, functions; and binary relations - Computer Journal, Vol.24,1, Feb.81.

H.Ellis — Analysing Business Information needs. - Proc.Data Analysis Conference 1978, University of Loughborough.

P.P.Chen — The entity-relationship model - ACM TODS

I.G.Macdonald & I.R.Palmer - System Development in a Shared Data Environment: The D2S2 Methodology - Information Systems Design Methodologies: A Comparative Review, Ed.T.W.Olle, North Holland (1982)

E.Tozer — Database Systems Analysis and Design - Software Sciences (1976)